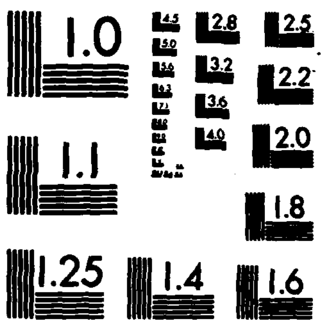
AD-A146 337 AN INPUT TRANSLATOR FOR A COMPUTER-AIDED DESIGN SYSTEM 1/2
(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA T H CARSON
JUN 84

UNCLASSIFIED F/G 9/2 NL


1.0
4.5
5.0
5.6
6.3
7.1
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN INPUT TRANSLATOR FOR A
COMPUTER-AIDED DESIGN SYSTEM

by

Thomas H. Carson

June 1984

Thesis Advisor: Alan A. Ross

Approved for public release; distribution unlimited

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A146337	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) An Input Translator for a Computer-Aided Design System		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June, 1984
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Thomas H. Carson		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE June, 1984
		13. NUMBER OF PAGES 120
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer-Aided Design, Input Translator, CSDL (Computer System Design Language), CSDE (Computer System Design Environment)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The purpose of this thesis is to design and implement the input translator for the Computer System Design Environment, which is a computer-aided design system. The Computer System Design Environment is used to design real time controllers for a variety of purposes. The input translator will take an input, which has been developed in the prescribed language, CSDL, and with the aid of a partial syntax-directed editor, (Continued)

ABSTRACT (Continued)

translate it into primitive list form. This form is used by the remainder of the system to select the best hardware and software components, as described in a set of realization libraries, to build the proposed controller.

An Input Translator for a Computer-Aided Design System

by

Thomas H. Carson
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1971

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1984

Author: [illegible]

Approved by: Alan A. Ross
Thesis Advisor

[illegible]
Second Reader

[illegible]
Chairman, Department of Computer Science

[illegible]
Dean of Information and Policy Sciences

Control System Design Language,

ABSTRACT

The purpose of this thesis is to design and implement the input translator for the Computer System Design Environment, which is a computer-aided design system. The Computer System Design Environment is used to design real time controllers for a variety of purposes. The input translator will take an input, which has been developed in the prescribed language, ~~CSDL,~~ and with the aid of a partial syntax-directed editor, translate it into primitive list form. This form is used by the remainder of the system to select the best hardware and software components, as described in a set of realization libraries, to build the proposed controller.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION AND BACKGROUND

A. COMPUTER-AIDED DESIGN

Computers, as design tools, are beginning to touch every facet of our lives. We can turn on our television sets to see an advertisement for an automobile with a computer being used to aerodynamically design the body style of the car. Architects are using computers to assist in modern drafting and architectural design techniques [Ref. 1]. The potential for automating the design of many other products exists and research in all areas of computer-aided design (CAD) is continuing at a prolific rate.

While research in the areas of artificial intelligence may lead us someday to a computer which can, using natural language understanding, solve problems reserved for only humans today, current technology limits us to those problems where the computer relies on human expert input for a knowledge base. A knowledge base developed by human experts is used by the computer to derive a design much faster than a human and reduce the concern over the complexity of the process. The computer can maintain a large data base of components from which it can pick to satisfy a criteria as described by the designer. While not really creative, this system allows the mixing and matching, automatically, of components to produce the best combination available. The advantages to be gained are a decrease in the time it takes to complete the design process and error free results, while leaving the human designer to concentrate on the desired specification.

One of the most important features of such a system is its accessibility. The user interface must be one that meets the needs of the designer while remaining within the bounds of current technology. The interface must be user-friendly to the greatest possible extent. It is this particular portion of the problem which has given rise to the most debate and brought forth the widest range of possible solutions.

B. COMPUTER SYSTEM DESIGN ENVIRONMENT

The Computer System Design Environment (CSDE), under development at the Naval Postgraduate School in Monterey California, is one such computer-aided design system. It is based on the research contained in LtCol. Alan Ross's doctoral dissertation [Ref. 2]. Ross's work is an expansion and realization of the research conducted by M. N. Matelan [Ref. 3]. The first CSDE implementation is one in which real-time controllers (microprocessers) are designed based on a realization library (knowledge base) of current micro-processer technology. The system creates the problem statement in a syntax-directed editor, translates it into an intermediate form, selects a microprocessor realization from the library and generates the software and hardware component descriptions to implement the design. The components are used to select a processor volume or implementation. The volume checked to see if the timing constraints, set forth by the designer, can be achieved. If so, the monitor is generated and the output is formatted. The monitor produces the software and ancillary hardware to realize the correct strategy. If the timing constraints cannot be met, a new volume must be chosen and tested. The CSDE gives a designer the tools to derive the appropriate components that make up the controller, no matter what its task is to be.

Motivation and discussion of the CSDE are contained in [Refs. 2 , 3].

C. PREVIOUS WORK

The modules that make up the Computer System Design Environment are depicted in figure 1.1. Matelan described a Control System Design Language (CSDL) as the input language for this system [Ref. 3]. Using CSDL with a syntax-directed editor keeps the input details at a high-level of abstraction while completely describing the proposed design.

To fulfill the input requirements of block 1 in figure 1.1, a syntax-directed editor was designed and partially implemented as a result of Lt. Barbara Sherlock's thesis at the Naval Postgraduate School in 1983 [Ref. 4]. Lt. Sherlock's editor receives a high level input description of the problem from the designer, formats it and passes it to the input translator. This form is a combination of Matelin's CSDL and ADA, the Department of Defense sponsored design language. This language, as the basis for input to and output from the editor, follows the concept that the problem statement should not require the designer to be proficient in the details of a high level programming language. The translator, as its name suggests, translates the output from the editor into an intermediate form acceptable to the follow on CSDE processes. Its design and implementation are the subjects of this thesis. The optimizer and functional mapper(Blocks 3 and 4, Figure 1.1) exist as Fortran programs in the CSDE. The optimizer requires an 80 column format for its input which is a primitive list or the set of functions that the controller will perform, in an almost assembly-like language format. It is developed from the contingency and procedures sections of the design input statement. Hardware availability

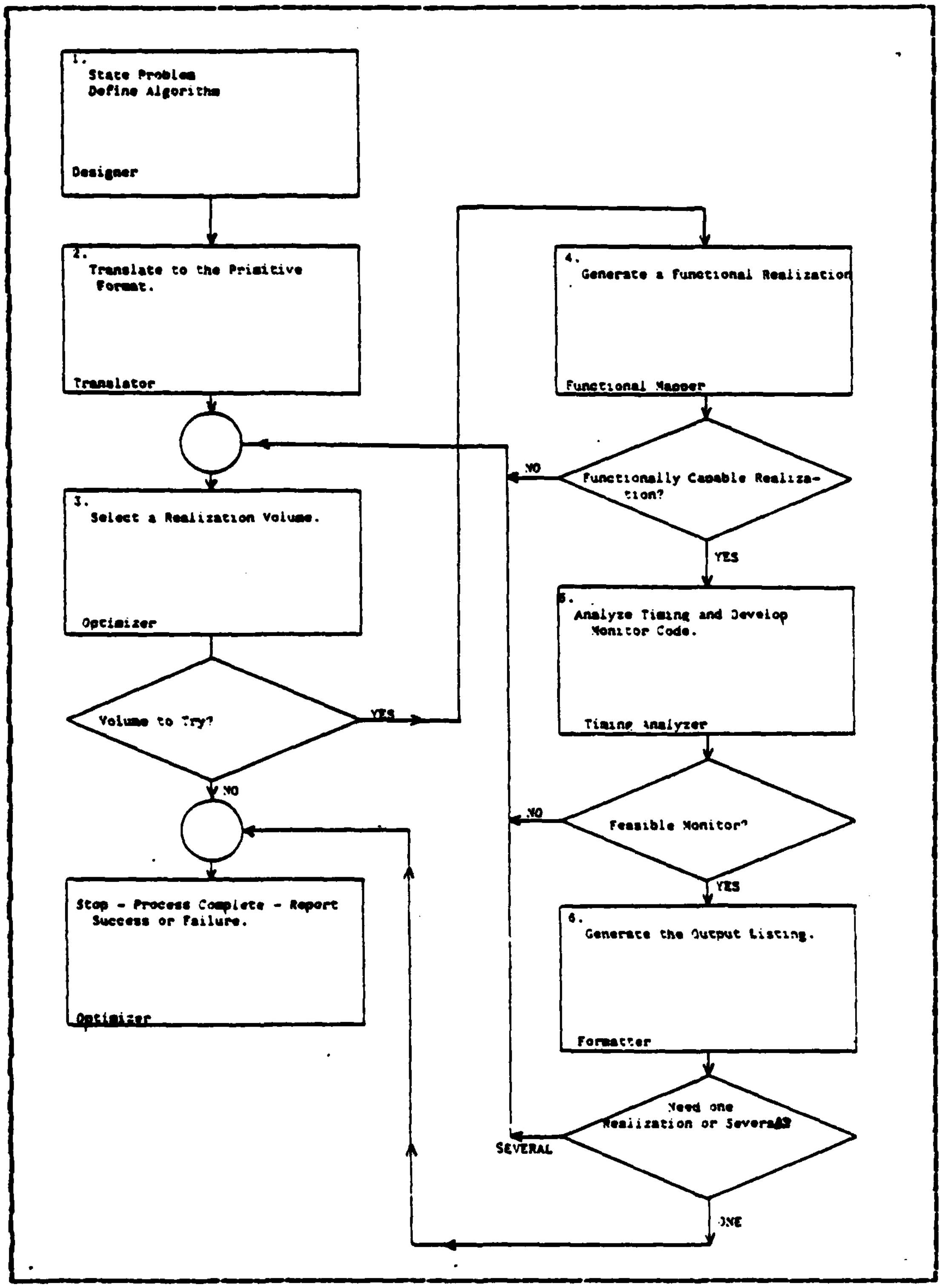

1. State Problem Define Algorithm
Designer
2. Translate to the Primitive Format.
Translator
3. Select a Realization Volume.
Optimizer
Volume to Try?
YES
NO
Stop - Process Complete - Report Success or Failure.
Optimizer
4. Generate a Functional Realization
Functional Mapper
Functionally Capable Realization?
NO
YES
5. Analyze Timing and Develop Monitor Code.
Timing Analyzer
Feasible Monitor?
NO
YES
6. Generate the Output Listing.
Formatter
Need one Realization or Several?
SEVERAL
ONE

Figure 1.1 Computer System Design Environment.

prescribed this format when the CSDE was originally implemented by LtCol. Ross. Work is currently in progress to install both the design input and the realization library in a relational database, thereby updating the optimizing and mapping processes [Ref. 5]. But the basis of the input to these sections of the system, which is the primitive list, will remain the same.

D. SCOPE OF THIS THESIS

The purpose of this thesis is to design and implement the input translator for the Computer System Design Environment. The translator must take the *problem statement*, a design for a real time microprocessor controller, and translate it into a list of primitives and a symbol table which can then be mapped across a realization library to determine the most feasible components. Appendix A is an example of a problem statement in the Computer System Design Language *developed* by Matelan. In addition, the translator must produce a timing table which can be used by the timing analyser to determine a feasible monitor to control the complete device [Ref. 2]. The timing table is developed using the contingency section of the problem statement which contains the timing requirements for the problem.

The next chapter of this thesis describes the design considerations for the translator and provides the detailed discussion of its requirements. The following chapters discuss the implementation and testing of the translator and the conclusions reached during the work. In addition, Appendice D includes the code for the translator with appropriate documentation.

II. TRANSLATOR DESIGN

A. DESIGN REQUIREMENTS AND CONSTRAINTS

As previously discussed, the input translator can be thought of as one of several modules in the CSDE system. In the CSDE hierarchy, it lies between the designer and the optimizer. If we consider, for a moment, the module as a black box, then we can better describe its function. The input to the module is a specification, written by the designer, for a real time controller. While Lt. Sherlock, in her design of the input editor [Ref. 4], decided to produce a pseudo-ADA specification language as the output of the editor, the translator being designed by this author will use the Control System Design Language. If the ADA ouput was to be adapted, this language would have to be formalized and a grammar produced that is capable of being parsed. In addition, the pseudo-ADA provides no real advantage, as the specification must be parsed and the same output produced no matter what the language. Therefore, with the additional knowledge that the editor is the subject of a current thesis project which will return to the CSDL output, the decision was made to write the translator for that CSDL. A partial example of the CSDL description is contained in figure 2.1.

The output from the translator consists of a primitive list, a symbol table, and an application timing table. The primitive list is intermediate code which reflects the requirements of the input while the symbol table contains all input variables and their attributes. The application timing table contains the contingencies with their related tasks and all supplied timing values from the problem definition. This table is used during the timing analysis.

```
CONTINGENCY LIST
  WHEN ALARM : 2MS,500US DO ALERT;
  EVERY 4MS : DO ENCODE;
  WHEN DATA_READY : 1300US DO SERIALIZE;

PROCEDURES

 FUNCTION DATA_READY:
  BINARY,1;
  SENSE (BUFFER);
  IF BUFFER /= OLDBUF THEN DATA_READY := 1 END IF;
 EXIT DATA_READY;
```

Figure 2.1 A Partial Example of CSDL.

The requirements, then, exist for the input language, CSDL, to be analyzed to determine what method of translation is to be employed. In addition, the required output must be standardized among the system modules so the proper semantics can be developed for the translation process. Each of these issues will be discussed in detail in the following sections.

B. CSDL- THE INPUT LANGUAGE

A translator accepts a source program, written in a source language, and transforms it into an object program [Ref. 6]. A source language designed for use in a computer aided design system and utilized in CSDE is the Control System Design Language (CSDL), the origin of which was previously discussed. It is composed of an alphabet whose individual elements are called tokens and a grammar which expresses the rules governing the legal classes of token strings. The tokens can be futher subdivided into terminals and nonterminals. Terminals are the letters of the allowed alphabet while the nonterminals are representations of strings in the language which increase the expressive power. A partial example of the production rules for CSDL is contained in figure 2.2. The syntax for CSDL, which includes the alphabet and grammar, is contained in Appendix B.

```
<WHEN DO> ::= <QUALIFICATION> WHEN <NAME>
              <EPISODE TIMING> DO <TASK LIST>

<TASK LIST> ::= <NAME> / <TASK LIST> THEN <NAME>

<NAME> ::= *ID* / *ID* ( <EXPR LIST> ) /
           *ID* *NUMBER* : *NUMBER*
```

Figure 2.2 An Example of CSDL Syntax.

Different types of translating devices accept different languages classifications. To narrow the choice of translator designs, it must be determined which classification fits CSDL. Neither Matelan nor Ross, in their early work on CSDE, included this description [Refs. 2 , 3]. So, a brief review of language classification will assist in this determination. Chomsky distinguished four general classes of grammars [Ref. 7]. Without turning this section into a text on language theory, these are, from the most general to the most specific: unrestricted, context-sensitive, context-free, and right-linear. Context-sensitive and context-free are subsets of the unrestricted class, while the right-linear and two other grammars related to the right-linear grammars, left-linear and regular, are all subsets of the context-free grammars. These classes allow us to define sentence recognizing machines which form the basis for translators. CSDL falls into the category of context-free grammars. This is the set which, in its production rules, has any string of terminals and nonterminals on the right-hand side of the production while the left-hand side is restricted to nonterminals only. The classes of right-linear, left-linear, and regular grammars restrict the order and appearances of terminals and nonterminals on each side of the production rules and CSDL does not fall into one of these categories. Context-sensitive grammars allow terminals as well as one nonterminal on the left-hand side of the production rules and,while CSDL does fit this category, the context-sensitive are a super-set of the context-free grammars, so this is not an issue when we try to develop the machines which can recognize CSDL.

Each of the phrase-structured grammar classes has an automaton associated with it. The right-linear grammars can be recognized and accepted by a finite-state automaton which consists of a finite set of states and a set of transitions between pairs of states. Each transition is associated with some terminal symbol. The context-sensitve grammars are recognized and accepted by a two-way, linear bounded automaton which is essentially a Turing machine whose tape cannot grow longer than the input string. And, finally, context-free grammars are recognized and accepted by a finite-state automaton controlling a push-down stack, with rules governing the operations on the stack [Ref. 6].

Matelan states that CSDL was created as a context-free grammar and inspection of the syntax contained in Appendix B confirms this [Ref. 3]. In order to recognize strings in the language and translate them into the presribed primitive-list format a finite-state automaton with a push-down stack will be developed.

C. PARSER ALTERNATIVES

It was shown above that CSDL is a context-free grammar and a push-down automata will be required as the recognizer for strings in the language. Additional properties of CSDL must be investigated to further define the problem of parsing. In a context-free grammar each nonterminal can be expanded into some terminal string independently of its neighbors, and its expanded string essentially "pushes aside" its neighbors without interfering with their order in any way [Ref. 6]. But we do impose some ordering rule for the selection of the next nonterminal to replace, in a sentential form for a canonical derivation. The most common rules are left-most and right-most. In a left-most derivation, the left-most nonterminal in each sentential form is selected for the next replacement and in a right-most, the right-most nonterminal is selected. Most common programming languages are easily parsed from left to right, but with difficulty from right to left. Futhermore, algebraic operations are usually performed from left to right, by convention, so it is the order that will be considered. A top-down parse of some sentence, scanning from left-to-right through the string, corresponds to a left-most derivation while a bottom-up parse works from a given sentence upward toward the start symbol, in a left-to-right manner. [Ref. 6].

There are some rules governing the use of these two types of parsers which affect the choice of one for use in recognizing CSDL. Top-down recognition with a look-ahead of k symbols is only possible on a subset of the context-free grammars called LL(k) grammars. Although it is not obvious whether a grammar is LL(k), there is one property which is relevant in this discussion. An LL(k) grammar has no left recursive nonterminals,i.e., a nonterminal A, such that A =>

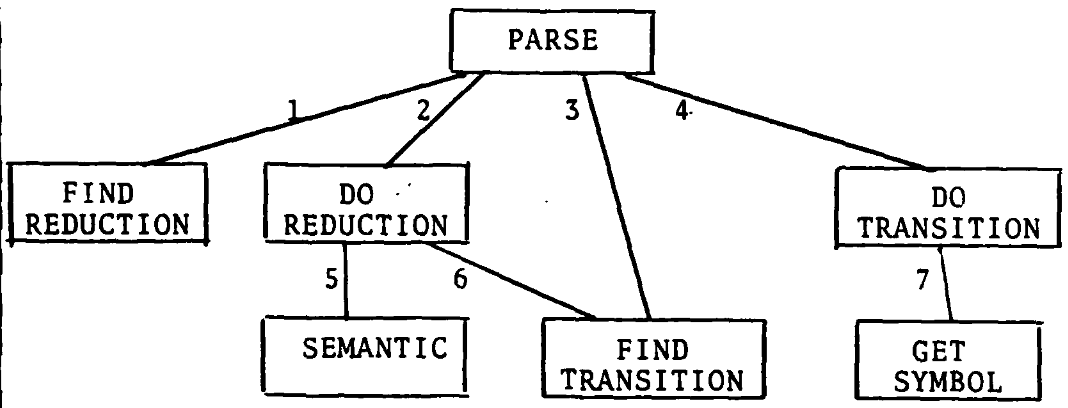
Aw for some w, a string in the language [Ref. 6]. It can be quickly determined by examining Production 17 in Appendix B that CSDL is left-recursive. In fact, it is full of recursion. There are algorithms for removing left-recursion in grammars and for a small grammar that would be the choice. But CSDL has 190 production rules and removing the extensive recursion would increase the grammar size an unacceptable amount. So top-down parsing will be discarded as a possible parsing method.

A bottom-up LR(k) parser is the other major type of recognizer under consideration. A grammar is said to be LR(k) if, for every derivation, the production A => x can be inferred by scanning ux and at most the first k symbols of v in the following derivation step: uAv => uxv . The major advantage of this method is that an LR(k) parser can be constructed for any context-free grammar. This would eliminate the necessity to remove the left-recursion from CSDL.

There is one other major advantage in choosing a bottom-up LR(1) parser automaton. An automatic parser generator can construct, using a computer, the language specific tables that control the operation of the automaton. The LR package from Lawrence Livermore Laboratory [Refs. 8 , 9] is such a system which constructs the tables. Having it available on the Vax 11/780 at the Naval Postgraduate School made the decision easy. It,also, has the advantages that the parsing routine is guaranteed to be correct, the CSDL grammar can be changed easily when necessary, and the resulting translator becomes simple and efficient. For the details as to how the package works see References 7 and 8.

D. PARSER STRUCTURE

The structure of the parser will follow the technique described by G.J. Myers in his book, Composite Structured Design [Ref. 10], and utilized in the design and implementation of an ADA pseudo-machine by Captain Alan Garlington in his thesis [Ref. 11]. The hierarchy for such a technique is



	IN	OUT
1	Current state and token	Production #
2	Production #, Stkptr	New state #, Stkptr
3	Current state and token	New state #
4	New state #, Stkptr	Stkptr, Current state Token description
5	Production #, Stkptr	-----------
6	Stack(stkptr) state, Left-hand side (production #)	New state #
7	----------	Token description

Figure 2.3 Parser Structure.

depicted in figure 2.3. The top module, PARSE, provides FINDREDUCTION with the current state and current look-ahead symbol. FINDREDUCTION returns a production number if any reduction exists. PARSE then calls DOREDUCTION and the sequence is repeated. If no reduction exists, PARSE calls FINDTRANSITION to see if any transitions exist. DOTRANSITION accomplishes the transition and the routine repeats until a final state is reached. If no transition exists, an error is detected and the routine either attempts to recover or halts depending on the severity of the error.

The parser maintains two stacks, one to store the next token and one to store the current state. When DOREDUCTION provides SEMANTIC with the production number, the components of the production have been placed on the stack and SEMANTIC can take the the proper action. This action could include adding a symbol to the symbol table with its appropriate parameters, calling an error routine,or nothing. After completing the semantic actions the items on the stack are removed by DOREDUCTION and the proper token replaces them. Various auxiliary procedures and error routines will be necessary to complete the translator. These procedures will include the input and output routines required to read the designers input and place the created primitives and symbol table in proper format.

The translator, thus, will take a CSDL description of the desired controller, check for errors, parse the input, and produce the primitive list , the symbol table, and application timing table. The implementation of the translator is discussed in the following chapter.

III. TRANSLATOR IMPLEMENTATION

A. LANGUAGE OF IMPLEMENTATION

The Computer System Design Environment was originally implemented in Fortran, with the system maintaining its data base on formatted punch cards for use in a batch environment. Feasible alternatives exist and have been investigated in subsequent research. CSDE is now installed in the Vax 11/780 at the Naval Postgraduate School, with interactive computing available in a variety of programming languages. However, it is still maintained, mainly, in Fortran. It has been shown to be conceptually feasible to represent the data base requirements of the CSDE by a relational model [Ref. 5]. Future intentions are to realize this concept on a data base management system such as Oracle, which is available on the Vax machine. The importance of this is that format, such as column numbers and location of data, and interface compatibility lose their importance and the relations between the data and the representations of the relations become major concerns.

An additional property required for the implementation language is maintainability. CSDL will not be static, as previously discussed. As hardware technology changes, the realization library will have to be updated and, accordingly, CSDL will have to modified. This will result in an update to the translator, reflecting, possibly, such changes as new primitives. A high level familiar programming language will ease the burden of maintenance for future users.

Pascal is a high level programming language which supports the above requirements and, therefore, was chosen over the possible alternatives. Pascal is familiar to most programmers and,in fact, is the "first language" taught to new Computer Science students at the Naval Postgraduate School. In addition, it is easily understood by programmers conversant with other block-structured languages.

The modularity available with Pascal's procedures, functions, and high level constructs will provide maintainability. Each major function in the parser will comprise a Pascal procedure, making the main body of the program simple. Also, a section to be modified or updated is self-contained, and can be seperately compiled and debugged following changes.

Perhaps the most important reason for selecting Pascal is the expected improvements in the CSDE system data base. When the libraries are placed in a relational database, and the designer's input must be mapped to a library in such a system, the data structures comprising the translator's outputs will require change. Pascal data structures are powerful and adaptable to a relational model, enabling this major modification to be completed without difficulty. Until such time as this occurs, the output will be of the form dictated by current system implementation. The details of this will follow.

B. TRANSLATOR INPUT

The input to the translator is the designer's requirements for a controller, written in the Control System Design Language, an example of which is contained in Appendix A. At this writing, a partial syntax-directed editor is under development which will create, based on the designer's ideas, the input in syntactically correct form. No conceptual basis for the storage or file format of the input has been previously discussed, so several assumptions were made to enable production of the translator. Since the program is to be written in Pascal and reside as a member of the larger system, the CSDE, a simple file containing the CSDL problem description, which is read by the translator, will be utilized as the method of input. The file will be a text file with the only formatting restrictions being those imposed by the syntax of CSDL, the language being parsed. While this method of input will, currently, require some hands on system manipulation during run time, it is envisioned that a system macro can easily be developed at a later date to automate the process.

It is intended that the input be provided to the translator in syntactically correct form. However, as mentioned above, the means for this is not yet implemented. As example problem statements and test cases have been generated to exercise the translator, a requirement has developed for syntax error detection. This requirement can be eliminated and the ensuing code removed upon completion of the editor.

C. PRINCIPAL PROCEDURES AND DATA STRUCTURES

The parser was summarized at a high level of abstraction in Chapter 2 of this thesis. This section will point out the important procedures and data structures employed by the translator. The supporting functions which complete the translator are described as they appear in the program in Appendix D.

The tables produced by the automatic parser generator, which control the operation of the parser, are placed in arrays. The array sizes were set in program constants and would be modified if a change to CSDL caused a modification to the tables. In addition, the symbol table, the state and lookahead stacks, and the temporary and constant lists are all implemented as arrays of records. Each record contains such information as the type and precision of the variable or constant and a pointer to the next record in the list. With the continuous manipulation of these data structures, such as pops and pushes on the stacks as well as the requirement for access to each member of each list, it was determined that this implementation allowed the maximum degree of flexibility. Also, the size of each data structure was described by a program constant, in order to improve maintenance. A limit might change in a situation where a controller design required a large number of input and output signals or internal variables, exceeding the maximum allowed for a stack.

Four sections conceptually comprise the translator program. The first is the initialization sequence, comprised of the procedure INITIALIZE and supporting functions. This section sets the initial values for all program variables and initializes the temporary and constant lists as well as the input symbol table to null values. It also establishes the SYMTABLE, which is a list containing all reserved words in CSDL, each located by a pointer. This table is used in the program to check each input token to determine whether it is a reserved word, identifier, or operator. Additionally, INITIALIZE includes the procedures for sending the generated primitive list, symbol table, and scratch pad to output files.

The next section is the actual parsing routine, comprised of the procedures PARSE, FINDREDUCTION, DOREDUCTION, FINDTRANSITION, DOTRANSITION, and their supporting procedures and functions. This sequence of code, essentially, repeats itself, looking at each input token, retrieved by GETSYM, and attempts to move through the required production until a final state is reached. The tokens are placed on the stack and if a reduction can be performed, control moves to the semantic portion of the program. If one cannot be done, the sequence finds the transition and continues movement through the production rule, getting the next input symbol, with the stack unchanged. If no transitions exist, an error is present in the input.

When a reduction number, which corresponds to a production number in the CSDL grammar, is found, the program sequences to the third main section which contains the semantic operations for the program. It includes the procedures SEMANTIC and SEMANTIC1. These are two large case statements which, for each production, do the proper stack operations and send the output information to the procedures in the initialization section to be formatted and filed. The semantic operations are called from within DOREDUCTION, but really comprise a seperate module within the structure of the translator.

The last section of the translator is the set of procedures comprising the error handling routines. This includes procedures PRINTERRORS, RECOVER, ERROR, and PRINTLINEERRORS. The parsing routine attempts to recover, using the procedure of the same name, from an error while documenting it to the user. If the parser cannot recover, the program will halt and print the complete list, in a file, of the errors noted prior to the crash. This error handling sequence can be eliminated from the translator, as previously noted, on completion of a syntax directed editor, which would ensure error-free input.

D. TRANSLATOR OUTPUT

1. Primitive List

The primary output of the translator is the primitive list, a sample of which is contained in figure 3.1. This list of primitives is similar to a set of macros with

```
P    1s.generated for:DATAA
P    2s.proc         (DATAA:)
P    3s.sensecond    (FLGA:1)
P    4s.eq           (@T01,FLGA,@C01:8,1,8)
P    5s.jmpf         (@T01,@01:8)
P    6s.assign       (DATAA,@C01:1,8)
P    7s.loc          (@01:)
P    8s.exitproc     (DATAA:)
```

Figure 3.1 A Sample of the Primitive List.

the operands and attributes corresponding to parameters. It is converted by the Optimizer module, in the CSDE, to the internal format required by succeeding modules [Ref. 2].

In the initial implementation of CSDE there was no front end, i.e., the modules for the designer input and translation to intermediate form did not exist. Therefore, the information required in each primitive was hand generated and formatted by column number so it could be inputed in batch form as a card image file. Because the information is still required in the same format, the output, containing the primitive list and called PRIMFILE, was set up in an identical manner. Each primitive is one line and spaces were added to emulate the blank columns on a punch card. The name of the primitive appears first (software primitives preceded by s., hardware by h.). This is followed by the operand list and selection list (if any) separated by a colon [Ref. 16].

The primitive list, also, contains the design criteria, preceded by d., which allows the designer to specify the order of consideration of the realization volumes. The generation of many realizations can also be requested, each of which is presented along with its chip count and power requirements. The final portion of the primitive list is the application timing table, with each line preceded by a beginning A. This table includes the timing constraints and associated requirements. Further details concerning the information present in each column in the sections of the primitive list are contained in Reference 2 and, therefore, will not be discussed here.

The primitive names, such as eq and mult, were chosen to correspond as closely as possible to the operation suggested, but the names in the realization volumes must be identical to the ones emitted by the translator for the CSDE to function. They are easily modified in the semantic portion of the program if required. However, new operations will dictate a modification to the CSDL grammar with additional productions and semantic rules.

2. Symbol Table

The concept of a symbol table, for use by succeding modules in the CSDE, is a result of this thesis. With the format required in the primitive list, it is difficult for the modules to look up, during the mapping, the attributes such as type, precision, value, and technology for controller variables and constants. It is thought that if this information were available in an easily read form, it would increase the speed of the realization process and raise the level of system efficiency. The functional mapper could read the symbol table first, and generate memory requirements for the controller prior to addressing the operations.

A symbol table can take as many forms as there are formats for a file. The data structure used to hold the information is also arbitrary. At the time of this writing, no decision has been made as to the ideal implementation. So, the information will be dumped into a file in the same format as the primitive list and can later be changed. This information includes all variables with their initial value and precision, all required memory locations, and all constants. In addition, the input and output ports with the expected signal names, technology, and precision are included for possible use. The code to format the symbol table can either be written in a seprate small routine to be installed as a system module, or could be added to the translator. Whichever the case, a decision in this matter may well come in time for the code to be included in the final version of the translator for this thesis, in which case it will be reflected in the example symbol table in the appendices.

3. Scratch Pad

A scratch pad file developed naturally as the translator was designed and implemented. Used initially for output, it significantly aided the debugging and verifying of the processes. Error routines, mentioned previously, send error diagnostics to this file. Traces of parser execution which were developed out of necessity as part of the debugging process also needed an output medium. Because of this, the scratch pad became a formal part of the translator. This file, TRANSLATE, is a text file, with no particular format, containing information which can be helpful to the user. If an error is detected in the input, the diagnostic, which traces the error, will appear here, with comments as to possible corrective action.

In addition, three toggles have been included which provide, if desired, 3 types of traces of program execution. TRACEPARSE will trace the parsing action, providing the transition and reduction numbers as the parser moves through the input. TRACETOK provides the input tokens, one at a time, as they are read. PRINTTABLE displays the controller symbol names with their attributes. These toggles are activated by including "--#" followed by a toggle name at the head of the inut file. More than one toggle per execution can be utilized, but the ensuing report becomes difficult to comprehend.

IV. TESTING AND VALIDATION

A. THEORY OF TESTING

The theory of software testing is a difficult problem and the subject of extensive research. Preliminary reports from this research indicate varying effectiveness. Dijkstra says that debugging can only show the presence of errors, but never their absence [Ref. 12]. It is commonly agreed that program testing cannot assure program "correctness" except under special circumstances. But the debate over Dijkstra's statement continues because others have developed theorems which challange his reasoning [Ref. 13].

The most important factor in testing is to have a well-understood goal for the testing process. In the case of the input translator, proving correctness was not at issue. The algorithm for the parser is well proven to be a correct one [Ref. 9], so the detection of bugs in the program was the goal. Methods of testing include top-down versus bottom-up, static versus dynamic, white box versus black box, and other less known systematic approaches. In bottom-up testing, the idea is to build the program with proven (bug free) components, while top-down begins with tests of the highest level using stubs to simulate the activity of the lower level modules. Static testing attempts to demonstrate the truth of an allegation, i.e., it roughly corresponds to bench-testing a power-driven device without applying power, and dynamic testing seeks to exercise a program in a controlled and systematic way. The white box testing approach is, knowing any part of the sofware system is present for some specific reason, then relating each piece of a software system to the requirement it fulfills. The black box method is an extensive testing approach which attempts to demonstrate the presence of function by concentrating on the exterior specifications of the software system [Ref. 14].

Just these brief examples illustrate the fact that software testing is not a well-defined science. The primary reason for this is that the concern for software reliability is relatively new. Only in the past decade has any notable effort been expended in understanding how a program can be proven correct or demonstrated to be reliable.

In selecting possible approaches to establish the reliabilty of the input translator, a combination of methods was determined to be the best. As previously noted, program correctness is not the objective. Also, the individual modules were completely debugged as they were built. So the problem reduces to ensuring the function between the input and output is such that the correct output is realized for each possible input. This is a black box methodology, but, since the program is to be exercised as a whole, the concept of dynamic testing also applies. The following section discusses the results of this testing and validation.

B. TEST RESULTS

Functional testing, a form of dynamic testing, involves the testing of a system over each of the different possible classes of input, the testing of each function implemented by the system, and the generation of test output in each of the possible output classes [Ref. 15]. This is the methodology that was employed in the testing of the input translator.

The CSDL input is divided into 5 parts which are IDENTIFICATION, DESIGN CRITERIA, ENVIRONMENT, PROCEDURES, and CONTINGENCIES. Each of these sections was examined in detail to determine the finite set of permutations in structure and content, as set forth by the CSDL grammar.

These possibilities formed the basis for test cases to be used in exercising the input translator. Due to its complexity, the PROCEDURES section received the most attention, but each part is discussed, in terms of the test results, below.

The IDENTIFICATION section, an example of which is contained in figure 4.1, consists of 3 character strings

```
IDENTIFICATION
  DESIGNER: "Thomas H. Carson"
  DATE: "10 April 1984"
  PROJECT: "Start Malfunction Controller"
```

Figure 4.1 IDENTIFICATION Section of the Input.

which make up a portion of the documentation for the system. The strings are not parsed by the input translator, just simply read and ignored. This section is optional, as are they all, and the parser performs no other action on it.

The DESIGN CRITERIA, an example of which is contained in figure 4.2, allows the the user to specify the metric and number of monitors and volumes to be employed in the mapping process, the next module in the CSDE system. The metric is one of 3 choices, all character strings, while the monitors

```
DESIGN CRITERIA
  METRIC FIRST;
  VOLUMES 1;
  MONITORS 1;
```

Figure 4.2 DESIGN CRITERIA Section of the Input.

and volumes are integer values. The parser reads each of these and checks for correctness in terms of value. It then reformats the information and places it in a special portion

```
3d:FIRST      :        1:        1:
```

Figure 4.3 Primitive List Form of the DESIGN CRITERIA.

of the primitive list. The possible alternatives were exercised and the corresponding correct output was generated, an example of which is contained in figure 4.3.

The ENVIRONMENT section, an example of which is contained in figure 4.4, contains the variable declarations for the input. It has 4 parts which are: procedure declarations, input signals, output signals, and duplex signals. While each of the above is optional, the controller will have to sense at least one input and emit one output to have some function. Each declaration can have its name, structure, precision, initial value, or the technology associated

```
ENVIRONMENT
  INPUT: RPM,8,TTL; FIRE_SENSE,1,TTL;
         OIL_PRES,8,TTL; END INPUT;
  OUTPUT: SD1,1,TTL; SD2,1,TTL;
          FIRE_EXT,1,TTL; END OUTPUT;
  ARITHMETIC: STAGFLG,1; END ARITHMENTIC;
```

Figure 4.4 ENVIRONMENT Section of the Input.

with it. The type of declaration decides which and how many of the attributes each variable will have. For the internal program variables, in the ARITHMETIC section, the translator generates a system variable primitive and for each of the signals it produces the associated software primitive. Again, while the translator is parsing the structure of the input, the real work done is reformating the names and their associated attributes into the appropriate primitive. No errors could be detected in exercising the program over this

```
4s.inputport (RPM,TTL:8)
5s.inputport (FIRE_SENSE,TTL:1)
6s.inputport (OIL_PRES,TTL:8)
7s.outputport(FIRE_EXT,TTL:1)
8s.var       (STAGFLG:1,0)
```

Figure 4.5 Primitive List Form of the Input.

portion of the input. An example of the output generated by the above example is contained in figure 4.5.

The PROCEDURES section contains the functions and tasks which establish the purpose of the controller being realized. The differences between functions and tasks are: functions are allowed only one basic statement and return a value while tasks allow multiple statements and perform a job. The key to both is the basic statement which is one of several types seen in most programming languages. The alternatives are: if-then, while-do, for loop, assignment, data input, data output, perform task, and wait. The only ones that might not be familiar are the "perform task" and "wait". "Perform task" allows for nested procedures and the "wait" statement causes the program to suspend itself for a prescribed period of time. An example of a task is contained in figure 4.6.

```
TASK OVRSPD;
 IF START SWIT THEN
  SENSE (RPM);
  SD5 := 0;
  IF RPM > 74 THEN SD5 := 1; END IF;
  ISSUE (SD5);
 END IF;
END OVRSPD;
```

Figure 4.6   PROCEDURES Section Input Example.

This section is where the parser really does its work. It must parse each statement in the procedure and generate an appropriate section of primitives, including temporaries, assembly-language-like software primitives and labels, which fulfill the intent of the statement. Each of the basic statements was exercised through the translator without any error detection, but because of the increased complexity of this section , 100% reliability cannot be confirmed. To do so would require a technique such as path testing. This requires that every logical path through a program be tested at least once. Another possibility is to construct test data which causes each branch in the program to be traversed [Ref. 15]. Alogorithms for such testing exist, but the problems with each are the excessive time, CPU service, and output verification required. Therefore, complete path or branch testing was not attempted. However, the author's confidence in the correctness, after the testing that was conducted, is 100%. An output produced from the above input example is contained in figure 4.7.

The CONTINGENCY LIST section, an example of which is contained in figure 4.8, sets up the flow and timing for the controller by establishing how often each procedure should be executed. There are 4 types of statements allowed in this section: when-do, at time, simple do, and the every.

```
106t.generated for:OVRSPD
107s.proc       (OVRSPD:)
108s.jmpf       (START SWIT,@12:1)
109s.sensecond  (RPM:8)
110s.gt         (@T01,RPM,@C07:8,8,8)
111s.jmpf       (@T01,@13:8)
112s.assign     (SD5,@C01:1,8)
113s.loc        (@13:)
114s.issuevent  (SD5:1)
115s.loc        (@12:)
116s.exitproc   (OVRSPD:)
```

Figure 4.7 Primitive List Form of the PROCEDURES Section.

While the parsing action for these statements, basically just a reformating routine, is simple, problems developed in

```
CONTINGENCY LIST
 WHEN RESET SWIT:100MS DO INIT;
 EVERY 1000MS DO CLOCK;
 EVERY 100MS DO OVRSPD;
```

Figure 4.8 CONTINGENCY LIST Input Example.

determining what the format in the primitive list should be. For consistancy, each was treated the same with blanks left in the columns in the "simple do", "every", and "at time" statements where a contingency name appears in the "when-do" statement. Examination of the example output in figure 4.9 and its comparison with the input example above will clarify this point. Alternative entries, such as the word "each", to replace the blanks are under consideration. The desision, based on the requirements of the functional mapper, will not occur prior to the submission of this thesis. Therefore, it is not possible to establish complete

```
A   1 :RESET_SWIT:INIT      :MS: 100,   0,   0,   0,   0
A   2 :           :CLOCK     :MS:1000,   0,   0,   0,   0
A   3 :           :OVRSPD    :MS: 100,   0,   0,   0,   0
```

Figure 4.9 Primitive List Form of the CONTINGENCY LIST.

correctness in this section. The program does act according to its given requirements but the possiblity exists for these to change and, at that time, the section will have to be reverified.

Only the primitive list has been discussed above as program output. The translator also generates a symbol table and a scratch pad. Since the symbol table is unformatted, testing established only that the required information was present. The scratch pad is not a functional member of the CSDE and, therefore, was not tested. The error-checking routines within the translator were tested in so far as they were used in creating correct input file for the testing described above. This was considered adequate due to the impending completion of a syntax-directed editor for composing the CSDL input for the translator.

The input translator was built bottom-up in modular form. This methodology and the use of the automatically generated driver for the parsing routines were significant reasons for the relatively error-free results obtained during the testing of the translator as a whole.

V. CONCLUSIONS AND RECOMMENDATIONS

A. PROGRAM MAINTENANCE

The use of the automatic parser generator in providing the control tables for the translator allows the maximum degree of flexibility for program maintenance. The resulting main program is modular and space efficient, with anticipated changes, such as new emissions from the semantic routines, easily included in the system.

The most obvious portion of the system which will undergo modification is the CSDL grammar. Matelan states there is no capability in CSDL for notational extensibility, nor should there be [Ref. 16]. He points out it is doubtful that the average user can design extensions that will cause less harm than good and the provision of a good macro capability and extendable function/task libraries are preferred. This author disagrees. The advantage of using the parser generator is that a new set of tables can be produced for a modified CSDL with virtually no disturbance to the translator. The changes to the grammar must be consistent with the rules governing LL(1) grammars, but such modifications as the inclusion of new tokens or reserved words is within the capability of advanced program language students. In addition, to keep CSDL static, using the function/task libraries to realize new primitive operations as they become technologically feasible, only results in a less efficient controller. It could lead, in the worst case, to the inability to utilize the latest hardware technology available for controller design. This was most certainly not the intent.

One particular change to CSDL needs immediate attention. Many controllers require an analog input or output signal. Matelan, in his work, makes the assumption that analog information must be converted to a digital signal before it reaches the interface [Ref. 16]. If we have the ability to modify CSDL, this is no longer a complex issue. One possibility is to add a new input/output signal type with the CSDE calling for an A/D-D/A converter when this type is mapped to a library.

B. RECOMMENDATIONS FOR CSDE

The flow of information between modules in CSDE, prior to the impending completion of the designer and translator modules, is consistent. Since the system was implemented in Fortran and resides on one machine, there is no problem. But with the completion of the translator, the subject of this thesis, and the designer module having the same time schedule, the problem of interfaces becomes significant. The translator is written in Pascal and, while the ouput is a simple text file, the information flow will not be as it should. In addition, the designer module is being written in the "C" programming language which will generate, possibly, further interface problems. It is therefore recommended that the CSDE system be incorporated, as quickly as possible, into a data base management system such as Oracle. This was previously discussed in Chapter 3 of this thesis and the design of the data base was the subject of earlier research [Ref. 5]. This research pointed out that this redesign of CSDE should help streamline the operation and eliminate any complex programming schemes that were built out of necessity in earlier work. This concept will eliminate, completely, the interface problems, as the information will be input to and accessible from the database as it passes from one module to the next. An overall improvement in system documentation, efficiency, and usability should result while, at the same time, allowing each module to maintain its individuality and function.

C. SUMMARY

In completing the design and implementation of the input translator, the objective of this thesis has been accomplished. The translator has been tested and fufills the function required of the module in the Computer System Design Environment [Ref. 2]. The additional features added to the translator include an option to monitor the execution sequence in a variety of tracing modes and extensive error checking. While not considered integral parts of the program, these features were useful in its design and testing. As the CSDE evolves, these features might become superfluous, at which time they could be eliminated. But the information provided by the features should be considered prior to that decision.

The design methodology employed in the production of the translator was not inovative, but it allowed the program to be simple and straightforward. The use of the automatic parser generator was a time saver and proved to be a tool which will allow for the ease of future program maintenance as no other could. In one sense, we could say, while generating a module for a computer aided design system, a computer aided programming tool was central in the development.

The translator will reside in the CSDE on the VAX 11/780 VMS operating system. It is a Pascal program with the source and object code available under the filename "CSDL". The user instructions have been fully covered in the previous sections of this thesis.

APPENDIX A
TRANSLATOR INPUT EXAMPLE

```
IDENTIFICATION

  DESIGNER: "Alan Ross"
  DATE: "12-23-83"
  PROJECT: "Dual Process Control Application"

DESIGN CRITERIA

  METRIC FIRST;
  VOLUMES 8;
  MONITORS 8;

ENVIRONMENT

  INPUT: CONSIN,8,TTL; CONST,8,TTL; FLGA,1,TTL;PINA,8,TTL;
         FLGB,1,TTL; PINB,8,TTL; END INPUT;

  OUTPUT: VA,8,TTL; VB,8,TTL; END OUTPUT;

  ARITHMETIC: KCA,8; KCB,8; CNT_B,8; ITIA,8; ITIB,8; AINT,8;
              TDA,8; TDB,8; BINT,8; VSA,8; VSB,8; BDIFF,8;
              PSA,8; PSB,8; CONPTT,8; EA,8; EB,8; KPIA,8;
              EA1,8; EA2,8; EB1,8; EB2,8; KPIB,8;
              END ARITHMETIC;

PROCEDURES

  FUNCTION DATA_A:
    BINARY,1;
    SENSE (FLGA);
    IF FLGA = 1 THEN DATA_A := 1; END IF;
  END DATA_A;

  FUNCTION DATA_B:
    BINARY,1;
  SENSE (FLGB);
  IF FLGB = 1 THEN DATA_B := 1; END IF;
END DATA_B;

FUNCTION BCNT:
  BINARY,1;
  IF CNT_B >= 4 THEN BCNT := 1; END IF;
END BCNT;

TASK AFIX;
  ARITHMETIC: ADIFF,8; END ARITHMETIC;
  SENSE (PINA);
  EA := PINA*KCA - PSA;
  ADIFF := (3*EA - 4*EA1 + EA2)*5;
  AINT := AINT + EA/KC;
  VA := VSA + KCA*(EA + ITIA*AINT + TDA*ADIFF);
  ISSUE (VA);
  DATA_A := 0;
  EA2 := EA1;
  EA1 := EA;
END AFIX;

TASK B_CALC;
  SENSE (PINB);
  EB := PINB*KCB - PSB;
  BDIFF := (3*EB - 4*EB1 + EB2)*10;
  BINT := BINT + EB/KCB;
  CNTB := CNTB + 1;
  DATA-B := 0;
END B_CALC;

TASK BFIX;
  CNTB := 0;
  VB := VSB + KCB*(EB + ITIB*BINT + TDB*BDIFF);
  ISSUE (VB);
END BFIX;
```

```
  FUNCTION CONFLG:
    BINARY,1;
    SENSE (CONSIN);
    IF CONSIN > 0 THEN CONFLG := 0; END IF;
  END CONFLG;

  TASK CHGCON;
    SENSE (CONST);
    IF CONPTT = 1 THEN KCA := CONST; END IF;
    IF CONPTT = 2 THEN ITIA := 1/CONST; END IF;
    IF CONPTT = 3 THEN TDA := CONST; END IF;
    IF CONPTT = 4 THEN VSA := CONST; END IF;
    IF CONPTT = 5 THEN PSA := CONST; END IF;
    IF CONPTT = 6 THEN AINT := CONST; END IF;
    IF CONPTT = 7 THEN KCB := CONST; END IF;
    IF CONPTT = 8 THEN ITIB := 1/CONST; END IF;
    IF CONPTT = 9 THEN TDB := CONST; END IF;
    IF CONPTT = 10 THEN VSB := CONST; END IF;
    IF CONPTT = 11 THEN PSB := CONST; END IF;
    IF CONPTT = 12 THEN BINT := CONST; END IF;
  END CHGCON;

CONTINGENCY LIST

    WHEN DATA_A :100MS DO AFIX;

    WHEN DATA_B :50MS DO B_CALC;

    WHEN BCNT :100MS DO BFIX;

    WHEN CONFLG DO CHGCON;
```

APPENDIX B
FORMAL SYNTAX OF CSDL

TERMINALS	NONTERMINALS
1. (	84. <AOP>
2.)	85. <ARITHMETIC BODY>
3. *	86. <ARITHMETIC DEC>
4. **	87. <ARITHMETIC SPEC>
5. *ID*	88. <ASSIGNMENT STMT>
6. *NUMBER*	89. <AT TIME>
7. *STRING*	90. <B1>
8. +	91. <B2>
9. '	92. <BASIC STMT>
10. -	93. <BINARY BODY>
11. .	94. <BINARY DEC>
12. /	95. <BINARY PRECISION>
13. /=	96. <BINARY SPEC>
14. :	97. <CHARACTER REP LIST>
15. :=	98. <CHARACTER REP>
16. ;	99. <CODE DEC LIST>
17. <	100. <CODE DEC>
18. <=	101. <CODE ID>
19. =	102. <CODE SPEC>
20. ==	103. <CODE VAR SPEC>
21. =>	104. <CONTINGENCY DEF>
22. >	105. <CONTINGENCY LIST>
23. >=	106. <CONTROL SYSTEM DESIGN>
24. AND	107. <DATA INPUT>
25. ARITHMETIC	108. <DATA OUTPUT>

26. ASCII6
27. ASCII7
28. AT
29. BCD
30. BINARY
31. CODE
32. CONTINGENCY
33. COST
34. CRITERIA
35. DATE
36. DESIGN
37. DESIGNER
38. DO
39. DUPLEX
40. EBCIDIC
41. ECL
42. END
43. ENVIRONMENT
44. EVERY
45. FIRST
46. FOR
47. FROM
48. FUNCTION
49. H
50. IDENTIFICATION
51. IF
52. ITL
53. IN
54. INPUT
55. ISSUE
56. LIST
57. M
58. METRIC
59. MONITORS
60. MS
109. <DEC GP>
110. <DEC>
111. <DECIMAL PRECISION>
112. <DESIGN CRITERIA>
113. <DUPLEX SPEC>
114. <ENVIRONMENT SECTION>
115. <EPISODE TIMING>
116. <EVERY>
117. <EXPR LIST>
118. <EXPRESSION>
119. <EXP_2>
120. <EXP_3>
121. <EXP_4>
122. <FACTOR>
123. <FOR HEAD>
124. <FOR LOOP>
125. <FORMAL PARAM LIST>
126. <FUNCTION>
127. <FUNCTION_HEAD>
128. <ID LIST>
129. <ID SECTION>
130. <IF HEAD>
131. <IF THEN>
132. <INITIAL VALUE>
133. <INPUT SPEC>
134. <LABELED STMT>
135. <LEFT PART LIST>
136. <LIST BODY>
137. <MAX LOOP COUNT>
138. <METRIC>
139. <MOP>
140. <NAME>
141. <NU>
142. <NUMBER LIST>
143. <OUTPUT SPEC>

61. NOT
62. NS
63. OR
64. OUTPUT
65. POWER
66. PROCEDURES
67. PROJECT
68. S
69. SENSE
70. TASK
71. TERM
72. THEN
73. TO
74. TTL
75. UNTIL
76. US
77. VARIABLES
78. VOLUMES
79. WAIT
80. WHEN
81. WHILE
82. {
83. }

144. <PERFORM TASK>
145. <PERIOD>
146. <PI>
147. <PRIMARY>
148. <PROC DEC GP>
149. <PROC DEC>
150. <PROC GP>
151. <PROC SECTION>
152. <PROC>
153. <QUALIFICATION>
154. <RANK>
155. <RELATION>
156. <RELATIONAL OP>
157. <ROE>
158. <SIMPLE DO>
159. <SIMPLE EXP>
160. <STMT GP>
161. <STMT>
162. <STRUCTURE>
163. <SYSTEM GOAL SYMBOL>
164. <TASK LIST>
165. <TASK>
166. <TASK_HEAD>
167. <TECHNOLOGY>
168. <TERM>
169. <TIME MEASURE>
170. <TIME>
171. <TIMED BLOCK>
172. <TIMED_BLOCK_HEAD>
173. <TRANSMISSION BODY>
174. <TRANSMISSION DEC>
175. <WAIT UNTIL>
176. <WAIT>
177. <WAIT_HEAD>
178. <WHEN DO>

```
                                        179. <WHILE DO>
                                        180. <WHILE HEAD>
                                        181. <WHILE>
                                        182. <ZOPT PROC DEC GP>
```

THE PRODUCTIONS

```
1. <SYSTEM GOAL SYMBOL> ::= END <CONTROL SYSTEM DESIGN> END

2. <AOP> ::= 3.          / -

4. <MOP> ::= *
5.          / /

6. <RELATIONAL OP> ::= <
7.                   / <=
8.                   / =
9.                   / >
10.                  / >=
11.                  / /=

12. <PRIMARY> ::= *NUMBER*
13.             / *STRING*
14.             / <NAME>
15.             / ( <EXPRESSION> )

16. <FACTOR> ::= <PRIMARY>
17.            / <FACTOR> ** <PRIMARY>

18. <TERM> ::= <FACTOR>
19.          / <TERM> <MOP> <FACTOR>

20. <SIMPLE EXP> ::= <TERM>
21.                / <AOP> <TERM>
22.                / NOT TERM
23.                / <SIMPLE EXP> <AOP> <TERM>

24. <RELATION> ::= <SIMPLE EXP>
25.            / <SIMPLE EXP> <RELATIONAL OP> <SIMPLE EXP>
```

```
26. <EXP_4> ::= <RELATION>
27.             / <EXP_4> AND <RELATION>

28. <EXP_3> ::= <EXP_4>
29.             / <EXP_3> OR <EXP_4>

30. <EXP_2> ::= <EXP_3>
31.             / <EXP_2> => <EXP_3>

32. <EXPRESSION> ::= <EXP_2>
33.                  / <EXPRESSION> == <EXP_2>

34. <EXPR LIST> ::=
35.                  / <EXPRESSION>
36.                  / <EXPR LIST> , <EXPRESSION>

37. <IF THEN> ::= <IF HEAD> THEN <STMT GP> END IF

38. <IF HEAD> ::= IF <EXPRESSION>

39. <WHILE DO> ::= <WHILE HEAD> DO <STMT GP> END WHILE

40. <WHILE HEAD> ::= <WHILE> <EXPRESION> : <MAX LOOP COUNT>

41. <WHILE> ::= WHILE

42. <FOR LOOP> ::= <FOR HEAD> DO <STMT GP> END FOR

43. <FOR HEAD> ::= FOR *ID* FROM <EXPRESSION> TO
                <EXPRESSION> : <MAX LOOP COUNT>

44. <PERFORM TASK> ::= *ID*
45.                    / *ID* ( <EXPR LIST> : <ID LIST> )

46. <MAX LOOP COUNT> ::= *NUMBER*

47. <LEFT PART LIST> ::= <NAME> :=
48.                      / <LEFT PART LIST> <NAME> :=

49. <ASSIGNMENT STMT> ::= <LEFT PART LIST> <EXPRESSION>

50. <DATA INPUT> ::= SENSE ( <NAME> )
```

```
51. <DATA OUTPUT> ::= ISSUE ( <NAME> )

52. <TIME MEASURE> ::= H
53.                    / M
54.                    / S
55.                    / MS
56.                    / US
57.                    / NS

58. <PERIOD> ::= *NUMBER* <TIME MEASURE>

59. <TIME> ::= <PERIOD>
60.          / <TIME> <PERIOD>

61. <TIMED BLOCK> ::= <TIMED_BLOCK_HEAD> DO <STMT GP> END IN

62. <TIMED_BLOCK_HEAD> ::= IN <PERIOD>

63. <WAIT> ::= WAIT <PERIOD>
64.          / WAIT <EXPRESSION> : <PERIOD>

65. <WAIT UNTIL> ::= <WAIT_HEAD> <EXPRESSION> : <PERIOD>

66. <WAIT_HEAD> ::= WAIT UNTIL

67. <BASIC STMT> ::= <IF THEN>
68.                / <WHILE DO>
69.                / <FOR LOOP>
70.                / <PERFORM TASK>
71.                / <ASSIGNMENT STMT>
72.                / <DATA INPUT>
73.                / <DATA OUTPUT>
74.                / <TIMED BLOCK>
75.                / <WAIT>
76.                / <WAIT UNTIL>

77. <LABELED STMT> ::= *ID* : <BASIC STMT>

78. <STMT> ::= <BASIC STMT>
```

```
79.            / <LABELED STMT>

80. <STMT GP> ::= <STMT> ;
81.                 / <STMT GP> <STMT> ;

82. <PROC DEC> ::= <BINARY SPEC>
83.                  / <ARITHMETIC SPEC>
84.                  / <CODE SPEC>
85.                  / <CODE VAR SPEC>

86. <INPUT SPEC> ::= INPUT : <TRANSMISSION BODY> END INPUT

87. <OUTPUT SPEC> ::= OUTPUT : <TRANSMISSION BODY>
                       END OUTPUT

88. <DEC> ::= <PROC DEC>
89.            / <INPUT SPEC>
90.            / <OUTPUT SPEC>
91.            / <DUPLEX SPEC>

92. <DEC GP> ::= <DEC> ;
93.               / <DEC GP> <DEC> ;

94. <DUPLEX SPEC> ::= DUPLEX <TRANSMISSION BODY> END DUPLEX

95. <BINARY SPEC> ::= BINARY : <BINARY BODY> END BINARY

96. <ARITHMETIC SPEC> ::= ARITHMETIC : <ARITHMETIC BODY>
                           END ARITHMETIC

97. <TRANSMISSION BODY> ::= <TRANSMISSION DEC> ;
98.                  / <TRANSMISSION BODY> <TRANSMISSION DEC> ;

99. <TRANSMISSION DEC> ::= *ID* , <BINARY PRECISION> ,
                            <TECHNOLOGY>

100. <BINARY BODY> ::= <BINARY DEC> ;
101.                    / <BINARY BODY> <BINARY DEC> ;

102. <BINARY DEC> ::= *ID* <STRUCTURE> , <BINARY PRECISION>
                       <INITIAL VALUE>
```

```
103. <ARITHMETIC BODY> ::= <ARITHMETIC DEC> ;
104.                 / <ARITHMETIC BODY> <ARITHMETIC DEC> ;

105. <ARITHMETIC DEC> ::= *ID* <STRUCTURE> ,
             <DECIMAL PRECISION> <INITIAL VALUE>

106. <STRUCTURE> ::=
107.                / ( <NUMBER LIST> )

108. <NUMBER LIST> ::= *NUMBER*
109.                 / <NUMBER LIST> , *NUMBER*

110. <BINARY PRECISION> ::= *NUMBER*

111. <DECIMAL PRECISION> ::= *NUMBER*

112. <INITIAL VALUE> ::=
113.                  / , *NUMBER*

114. <TECHNOLOGY> ::= TTL
115.                / ECL
116.                / ITL

117. <CODE VAR SPEC> ::= CODE VARIABLES : <CODE DEC LIST>
                       END CODE VARIABLES

118. <CODE DEC LIST> ::= <CODE DEC> ;
119.                   / <CODE DEC LIST> <CODE DEC> ;

120. <CODE DEC> ::= *ID* : <CODE ID>

121. <CODE SPEC> ::= CODE : *ID* , <BINARY PRECISION> ;
                  <CHARACTER REP LIST> END CODE

122. <CHARACTER REP LIST> ::= <CHARACTER REP> ;
123.                /<CHARACTER REP LIST> <CHARACTER REP> ;

124. <CHARACTER REP> ::= *ID* : *NUMBER*

125. <CODE ID> ::= *ID*
126.             / ASCII6
```

```
127.                  / ASCII7
128.                  / EBCIDIC
129.                  / BCD

130. <ID LIST> ::=
131.                  / *ID*
132.                  / <ID LIST> , *ID*

133. <NAME> ::= *ID*
134.               / *ID* ( <EXPR LIST> )
135.               / *ID* { *NUMBER* : *NUMBER* }

136. <FORMAL PARAM LIST> ::=
137.                                  / ( <ID LIST> : <ID LIST> )

138. <PROC> ::= <TASK>
139.               / <FUNCTION>

140. <TASK> ::= <TASK_HEAD> ; <ZOPT PROC DEC GP> <STMT GP>
                END *ID*

141. <ZOPT PROC DEC GP> ::=
142.                              / <PROC DEC GP>

143. <PROC DEC GP> ::= <PROC DEC> ;
144.                        / <PROC DEC GP> <PROC DEC> ;

145. <TASK_HEAD> ::= TASK *ID* <FORMAL PARAM LIST>

146. <FUNCTION> ::= <FUNCTION_HEAD> ; <ZOPT PROC DEC GP>
                     <STMT> END *ID*

147. <FUNCTION_HEAD> ::= FUNCTION *ID* <FORMAL PARAM LIST> :
                 BINARY , <BINARY PRECISION> <INITIAL VALUE>
148.                         / FUNCTION *ID* <FORMAL PARAM LIST> :
              ARITHMETIC ,<DECIMAL PRECISION> <INITIAL VALUE>

149. <PROC GP> ::= <PROC> ;
150.                  / <PROC GP> <PROC> ;

151. <PROC SECTION> ::=
152.                        / PROCEDURES <PROC GP>

153. <ROE> ::= <PERIOD>

154. <B1> ::= <PERIOD>

155. <B2> ::= <PERIOD>

156. <RANK> ::= <NU>
157.              / <NU> . <PI>

158. <NU> ::= *NUMBER*

159. <PI> ::= *NUMBER*

160. <QUALIFICATION> ::=
161.                         / IF <EXPRESSION>

162. <EPISODE TIMING> ::=
163.                          / : <ROE>
164.                          / : <ROE> , <B1>
165.                          / : <ROE> , <B1> , <B2>
166.                          / : <ROE> , <B1> , <B2> , <RANK>

167. <WHEN DO> ::= <QUALIFICATION> WHEN <NAME>
                    <EPISODE TIMING> DO <TASK LIST>

168. <SIMPLE DO> ::= <QUALIFICATION> DO <TASK LIST> <RANK>

169. <EVERY> ::= <QUALIFICATION> EVERY <ROE> DO <TASK LIST>

170. <AT TIME> ::= <QUALIFICATION> AT <TIME> DO <TASK LIST>

171. <TASK LIST> ::= <NAME>
172.                     / <TASK LIST> THEN <NAME>

173. <CONTINGENCY DEF> ::= <WHEN DO>
174.                           / <SIMPLE DO>
175.                           / <EVERY>
176.                           / <AT TIME>

177. <LIST BODY> ::= <CONTINGENCY DEF> ;
```

```
178.                  / <LIST BODY> <CONTINGENCY DEF> ;

179. <CONTINGENCY LIST> ::=
180.                            / CONTINGENCY LIST <LIST BODY>

181. <DESIGN CRITERIA> ::=
182.                       / DESIGN CRITERIA METRIC <METRIC> ;
          VOLUMES <NUMBER LIST> ; MONITORS <NUMBER LIST> ;

183. <METRIC> ::= FIRST
184.             / COST
185.             / POWER

186. <ENVIRONMENT SECTION> ::=
187.                               / ENVIRONMENT <DEC GP>

188. <ID SECTION> ::=
189.                   / IDENTIFICATION DESIGNER : *STRING*
           DATE : *STRING* PROJECT : *STRING*

190. <CONTROL SYSTEM DESIGN> ::= <ID SECTION>
      <DESIGN CRITERIA> <ENVIRONMENT SECTION> <PROC SECTION>
      <CONTINGENCY LIST>
```

APPENDIX C

PRIMITIVE LIST

```
P   1t.generated for:   SYSTEM              ********************
P   2s.MAIN       ()
P   3d:FIRST      :8,8,0,0,0,0,0,           0:          0:
P   4s.inputport (CONSIN,TTL:8)
P   5s.inputport (CONST,TTL:8)
P   6s.inputport (FLGA,TTL:1)
P   7s.inputport (PINA,TTL:8)
P   8s.inputport (FLGB,TTL:1)
P   9s.inputport (PINB,TTL:8)
P  10s.outputport(VA,TTL:8)
P  11s.outputport(VB,TTL:8)
P  12s.var        (KCA:8,0)
P  13s.var        (KCB:8,0)
P  14s.var        (CNTB:8,0)
P  15s.var        (ITIA:8,0)
P  16s.var        (ITIB:8,0)
P  17s.var        (AINT:8,0)
P  18s.var        (TDA:8,0)
P  19s.var        (TDB:8,0)
P  20s.var        (BINT:8,0)
P  21s.var        (VSA:8,0)
P  22s.var        (VSB:8,0)
P  23s.var        (BDIFF:8,0)
P  24s.var        (PSA:8,0)
P  25s.var        (PSB:8,0)
P  26s.var        (CONPTT:8,0)
P  27s.var        (EA:8,0)
P  28s.var        (EB:8,0)
P  29s.var        (KPIA:8,0)
P  30s.var        (EA1:8,0)
P  31s.var        (EA2:8,0)
P  32s.var        (EB1:8,0)
P  33s.var        (EB2:8,0)
P  34s.var        (KPIB:8,0)
P  35t.generated for:DATAA                  ********************
P  36s.proc       (DATAA:)
P  37s.sensecond (FLGA:1)
P  38s.eq         (@T01,FLGA,@C01:8,1,8)
P  39s.jmpf       (@T01,@01:8)
P  40s.assign     (DATAA,@C01:1,8)
P  41s.loc        (@01:)
P  42s.exitproc   (DATAA:)
P  43t.generated for:DATAB                  ********************
P  44s.proc       (DATAB:)
P  45s.sensecond (FLGB:1)
P  46s.eq         (@T01,FLGB,@C01:8,1,8)
P  47s.jmpf       (@T01,@02:8)
P  48s.assign     (DATAB,@C01:1,8)
P  49s.loc        (@02:)
P  50s.exitproc   (DATAB:)
P  51t.generated for:BCNT                   ********************
P  52s.proc       (BCNT:)
P  53s.ge         (@T01,CNTB,@C02:8,8,8)
```

```
P  54s.jmpf      (@T01,@03:8)
P  55s.assign    (BCNT,@C01:1,8)
P  56s.loc       (@03:)
P  57s.exitproc  (BCNT:)
P  58t.generated for:AFIX                    ********************
P  59s.proc      (AFIX:)
P  60s.var       (ADIFF:8,0)
P  61s.sensecond (PINA:8)
P  62s.mult      (@T01,PINA,KCA:8,8,8)
P  63s.sub       (@T01,@T01,PSA:8,8,8)
P  64s.assign    (EA,@T01:8,8)
P  65s.mult      (@T01,@C03,EA:8,8,8)
P  66s.mult      (@T02,@C02,EA1:8,8,8)
P  67s.sub       (@T01,@T01,@T02:8,8,8)
P  68s.add       (@T01,@T01,EA2:8,8,8)
P  69s.mult      (@T01,@T01,@C04:8,8,8)
P  70s.assign    (ADIFF,@T01:8,8)
P  71s.divide    (@T01,EA,KCA:8,8,8)
P  72s.add       (@T01,AINT,@T01:8,8,8)
P  73s.assign    (AINT,@T01:8,8)
P  74s.mult      (@T01,ITIA,AINT:8,8,8)
P  75s.add       (@T01,EA,@T01:8,8,8)
P  76s.mult      (@T02,TDA,ADIFF:8,8,8)
P  77s.add       (@T01,@T01,@T02:8,8,8)
P  78s.mult      (@T01,KCA,@T01:8,8,8)
P  79s.add       (@T01,VSA,@T01:8,8,8)
P  80s.assign    (VA,@T01:8,8)
P  81s.issuevent (VA:8)
P  82s.assign    (DATAA,@C05:1,8)
P  83s.assign    (EA2,EA1:8,8)
P  84s.assign    (EA1,EA:8,8)
P  85s.exitproc  (AFIX:)
P  86t.generated for:BCALC                   ********************
P  87s.proc      (BCALC:)
P  88s.sensecond (PINB:8)
P  89s.mult      (@T01,PINB,KCB:8,8,8)
P  90s.sub       (@T01,@T01,PSB:8,8,8)
P  91s.assign    (EB,@T01:8,8)
P  92s.mult      (@T01,@C03,EB:8,8,8)
P  93s.mult      (@T02,@C02,EB1:8,8,8)
P  94s.sub       (@T01,@T01,@T02:8,8,8)
P  95s.add       (@T01,@T01,EB2:8,8,8)
P  96s.mult      (@T01,@T01,@C06:8,8,8)
P  97s.assign    (BDIFF,@T01:8,8)
P  98s.divide    (@T01,EB,KCB:8,8,8)
P  99s.add       (@T01,BINT,@T01:8,8,8)
P 100s.assign    (BINT,@T01:8,8)
P 101s.add       (@T01,CNTB,@C01:8,8,8)
P 102s.assign    (CNTB,@T01:8,8)
P 103s.assign    (DATAB,@C05:1,8)
P 104s.exitproc  (BCALC:)
P 105t.generated for:BFIX                    ********************
P 106s.proc      (BFIX:)
```

```
P 107s.assign    (CNTB,@C05:8,8)
P 108s.mult      (@T01,ITIB,BINT:8,8,8)
P 109s.add       (@T01,EB,@T01:8,8,8)
P 110s.mult      (@T02,TDB,BDIFF:8,8,8)
P 111s.add       (@T01,@T01,@T02:8,8,8)
P 112s.mult      (@T01,KCB,@T01:8,8,8)
P 113s.add       (@T01,VSB,@T01:8,8,8)
P 114s.assign    (VB,@T01:8,8)
P 115s.issuevent (VB:8)
P 116s.exitproc  (BFIX:)
P 117t.generated for:CONFLG               **********************
P 118s.proc      (CONFLG:)
P 119s.sensecond (CONSIN:8)
P 120s.gt        (@T01,CONSIN,@C05:8,8,8)
P 121s.jmpf      (@T01,@04:8)
P 122s.assign    (CONFLG,@C05:1,8)
P 123s.loc       (@04:)
P 124s.exitproc  (CONFLG:)
P 125t.generated for:CHGCON               **********************
P 126s.proc      (CHGCON:)
P 127s.sensecond (CONST:8)
P 128s.eq        (@T01,CONPTT,@C01:8,8,8)
P 129s.jmpf      (@T01,@05:8)
P 130s.assign    (KCA,CONST:8,8)
P 131s.loc       (@05:)
P 132s.eq        (@T01,CONPTT,@C07:8,8,8)
P 133s.jmpf      (@T01,@06:8)
P 134s.divide    (@T01,@C01,CONST:8,8,8)
P 135s.assign    (ITIA,@T01:8,8)
P 136s.loc       (@06:)
P 137s.eq        (@T01,CONPTT,@C03:8,8,8)
P 138s.jmpf      (@T01,@07:8)
P 139s.assign    (TDA,CONST:8,8)
P 140s.loc       (@07:)
P 141s.eq        (@T01,CONPTT,@C02:8,8,8)
P 142s.jmpf      (@T01,@08:8)
P 143s.assign    (VSA,CONST:8,8)
P 144s.loc       (@08:)
P 145s.eq        (@T01,CONPTT,@C04:8,8,8)
P 146s.jmpf      (@T01,@09:8)
P 147s.assign    (PSA,CONST:8,8)
P 148s.loc       (@09:)
P 149s.eq        (@T01,CONPTT,@C08:8,8,8)
P 150s.jmpf      (@T01,@10:8)
P 151s.assign    (AINT,CONST:8,8)
P 152s.loc       (@10:)
P 153s.eq        (@T01,CONPTT,@C09:8,8,8)
P 154s.jmpf      (@T01,@11:8)
P 155s.assign    (KCB,CONST:8,8)
P 156s.loc       (@11:)
P 157s.eq        (@T01,CONPTT,@C10:8,8,8)
P 158s.jmpf      (@T01,@12:8)
P 159s.divide    (@T01,@C01,CONST:8,8,8)
```

```
P 160s.assign    (ITIB,@T01:8,8)
P 161s.loc       (@12:)
P 162s.eq        (@T01,CONPTT,@C11:8,8,8)
P 163s.jmpf      (@T01,@13:8)
P 164s.assign    (TDB,CONST:8,8)
P 165s.loc       (@13:)
P 166s.eq        (@T01,CONPTT,@C06:8,8,8)
P 167s.jmpf      (@T01,@14:8)
P 168s.assign    (VSB,CONST:8,8)
P 169s.loc       (@14:)
P 170s.eq        (@T01,CONPTT,@C12:8,8,8)
P 171s.jmpf      (@T01,@15:8)
P 172s.assign    (PSB,CONST:8,8)
P 173s.loc       (@15:)
P 174s.eq        (@T01,CONPTT,@C13:8,8,8)
P 175s.jmpf      (@T01,@16:8)
P 176s.assign    (BINT,CONST:8,8)
P 177s.loc       (@16:)
P 178s.exitproc  (CHGCON:)
A   1 :DATAA     :AFIX      :MS: 100.   0.   0.   0.   0
A   2 :DATAB     :BCALC     :MS:  50.   0.   0.   0.   0
A   3 :BCNT      :BFIX      :MS: 100.   0.   0.   0.   0
A   4 :CONFLG    :CHGCON    :MS:   0.   0.   0.   0.   0
P 179t.generated for:  SYSTEM           ********************
P 180s.cons      (@C01:1,8)
P 181s.cons      (@C02:4,8)
P 182s.cons      (@C03:3,8)
P 183s.cons      (@C04:5,8)
P 184s.cons      (@C05:0,8)
P 185s.cons      (@C06:10,8)
P 186s.cons      (@C07:2,8)
P 187s.cons      (@C08:6,8)
P 188s.cons      (@C09:7,8)
P 189s.cons      (@C10:8,8)
P 190s.cons      (@C11:9,8)
P 191s.cons      (@C12:11,8)
P 192s.cons      (@C13:12,8)
P 193s.var       (@T01:8)
P 194s.var       (@T02:8)
```

APPENDIX D

TRANSLATOR SOURCE LISTING

```
PROGRAM CSDL (DAT,INPUT,OUTPUT,PRIMFILE,TRANSLATE,SYMFILE);
 (*This program uses the output from the CSDE syntax-directed*)
 (*editor. The ouput is in standard CSDL form and is         *)
 (*translated into a primitive list, contained in PRIMFILE,  *)
 (*and the results of the parse in the OUTPUT file.          *)

LABEL 99;

CONST
 (* Constants generated by the translator as a result of the     *)
 (* CSDL syntax being fed through the automatic parser generator.*)

  FREDSIZE = 397;  (*ARRAY LENGTH LIMITS*)
  NSETSIZE = 205;
  LSETSIZE = 63;

  LSSIZE = 573;
  PRODSIZE = 205;
  FTRNSIZE = 397;
  TRANSIZE = 1161;
  ENTSIZE =396;
  LHSSIZE = 190;
  LENSIZE = 190;
  FSTATE = 7;
  VOCSIZE = 182;
  FIRSTRESWD = 24;
  LASTRESWD = 81;
  NUMTERMINALS = 83;

  MAXENT =182;  (*MAXIMUM VALUES OF ARRAY ELEMENTS*)
  MAXFRED = 206;
  MAXFTRN = 1162;
  MAXTRAN = 396;
  MAXNSET = 62;
  MAXPROD = 190;
  MAXLS = 81;
  MAXLSET = 574;
  MAXLEN = 11;
  MAXLHS = 182;

  (*PROGRAM CONSTANTS*)

  ENDTOK =42;       IDTOK = 5;        NUMTOK = 6;
  ARROWTOK = 21;    EQUIVALENCE = 20;PWRTOK = 4;
  BECOMESTOK = 15; NOTEQTOK = 13;   LESSEQTOK = 18;
  GTREQTOK = 23;    STRINGTOK = 7;


  MAXSTK = 40; (* SIZE OF STACK USED IN PARSER *)
  TABSIZE = 54; (* SIZE OF SYMBOL TABLE *)
  LINELENGTH = 120; (* MAX LENGTH OF INPUT LINES *)
  PAGESIZE = 53; (* NUMBER OF LINES PRINTED PER PAGE *)
  LINERRARRAYSIZE = 10; (* MAX NO OF ERRORS FLAGGED PER INPUT LINE +2 *)
  MAXSTRINGS = 100; (* MAX LENGTH OF STRINGHEAD ARRAY *)
  MAXSSTORE = 1000; (* MAX LENGTH OF STRINGSTORE ARRAY *)
  TEMPLISTMAX = 25; (* MAX LENGTH OF A UTILITY IN SEMANTIC *)
  MAXTEMPS = 20; (* MAX LENGTH OF A UTILITY IN SEMANTIC *)
  TPOOLBSIZE = 10; (* CONSTANT USED IN SEMANTIC *)
  MAXEVALSTACK = 20;  (* DEPTH OF ARTH EVAL STACK *)
  MAXOPS = 10; (* MAX NUMBER OF OPS HELD IN STORAGE FOR PRINTING *)
  MAXSELS = 10; (* MAX NUMBER OF SELS HELD IN STORAGE FOR PRINTING *)
  CSSMAX = 20; (* MAX LENGTH OF A UTILITY IN SEMANTIC *)


TYPE


  (*SYMBOL TABLE TYPES*)

  ALFA = VARYING[MAXSSTORE] OF CHAR;
  KINDS = (UNDEFINED,RESWD,BINARY,ARITHMETIC,WORD,CHAREP,
           TRANSDEC,TASK,FNCTN);
  EXPTYPES = (INT,REEL,BOOL,STNG,ERRORS);
  SYMPTR = SYMENTRY;
  SYMENTRY = RECORD
               SYMNAME : ALFA;
               LINK : SYMPTR;
               CASE KIND : KINDS OF
      (*1*)  RESWD : (KEY1 : INTEGER);
      (*2*)  BINARY : (PRECISION2, IVAL2 : INTEGER);
      (*3*)  ARITHMETIC : (PRECISION3, IVAL3 : INTEGER);
      (*4*)  WORD : (CODID4 : SYMPTR; STRINGPTR4 : INTEGER);
      (*5*)  CHAREP : (CODID5 : SYMPTR; IVAL5 : INTEGER);
      (*6*)  TRANSDEC : (TYPE6, TECHNOLOGY6 : SYMPTR;
                         PRECISION6 : INTEGER);
      (*7*)  TASK : (PARAMLIST7 : SYMPTR);
      (*8*)  FNCTN : (PARAMLIST8, TYPE8 : SYMPTR;
                      PRECISION8, IVAL8 : INTEGER);
             END;


    (*SEMANTIC TYPES*)

  SWITCHES = (TRACEPARSE,TRACETOK,PRINTTABLE);
  DESCRIPTOR = RECORD    (* DESCRIBES THE CURRENT TOKEN *)
                 SYMNAME,TMPNAME : ALFA;
                 LINEPOS,
                 INTVAL : INTEGER; (* INTEGER VAL OF CURRENT NUMBER *)
                 REALVAL : REAL; (* REAL VAL OF CURRENT NUMBER *)
                 CHARVAL : CHAR; (* VAL OF CURRENT CHAR LITERAL *)
                 SYMLOC : SYMPTR; (* SYMBOL LOC IN SYMBOL TABLE *)
                END;
  TAGTYPE = (QNUM,QNAME);


VAR
```

```

PRIMFILE : TEXT;
DAT : TEXT ;
TRANSLATE : TEXT;
SYMFILE : TEXT;





FRED : PACKED ARRAY[1..FREDSIZE] OF 1..MAXFRED;
NSET : PACKED ARRAY[1..NSETSIZE] OF 1..MAXNSET;
LSET : PACKED ARRAY[1..LSETSIZE] OF 1..MAXLSET;
LS   : PACKED ARRAY[1..LSSIZE]   OF 1..MAXLS;
PROD : PACKED ARRAY[1..PRODSIZE] OF 1..MAXPROD;



FTRN : PACKED ARRAY[1..FTRNSIZE] OF 1..MAXFTRN;
TRAN : PACKED ARRAY[1..TRANSIZE] OF 1..MAXTRAN;
ENT  : PACKED ARRAY[1..ENTSIZE]  OF 1..MAXENT;


LEN  : PACKED ARRAY[1..LENSIZE] OF 0..MAXLEN;
LHS  : PACKED ARRAY[1..LHSSIZE] OF 1..MAXLHS;


(* PROCEDURE PARSE *)

NEXTSYM, (* NEXT SYMBOL IN INPUT STREAM *)
NOWSTA, (* CURRENT STATE *)
REDUCTION, (* POSSIBLE REDUCTIONS *)
TRANSITION : INTEGER; (* POSSIBLE TRANSITIONS *)

(* STACK VARIABLES *)

STACK : ARRAY[1..MAXSTK] OF RECORD
    STATE : INTEGER; (* STATE STACK *)
    TOK : INTEGER; (* TOKEN STACK *)
    DES : DESCRIPTOR; (* TOKEN DESCRIPTOR *)
    EXPTYPE : EXPTYPES;
  END;
STKPTR : INTEGER; (* TOP OF STACK POINTER *)

(* ERROR HANDLING VARIABLES *)

ERRLIST : SET OF 1..58; (* COMPILATION ERROR LIST *)
LINERRORS : ARRAY[1..LINERRARRAYSIZE] OF RECORD
             ERRPOSITION, (* ERROR POSITION *)
             ERRNUM, (* WHICH ERROR *)
             STATE : INTEGER; (* PARSER STATE WHERE ERROR OCCURRED *)
            END;
LINERRPTR,  (* POINTER INTO LINERRORS *)
```

```
MAXLINERRORS : INTEGER;
PROGRAMERRFLAG,
OVERFLOWLOGGED : BOOLEAN;

(* PROCEDURE NEXTSYM *)

SPS : ARRAY[CHAR] OF INTEGER;
CC,                              (* POINTER TO CURRENT CHAR ON LINE *)
LL,                              (* LENGTH OF CURRENT INPUT SYMBOL *)
SOURCELINECOUNT,                 (* INPUT LINE COUNTER *)
PAGENUMBER,
PAGELINECOUNT : INTEGER;
CH : CHAR;                              (* NEXT CHAR IN LINE *)
LINE : ARRAY[1..LINELENGTH] OF CHAR;       (* INPUT LINE BUFFER *)
ZDATE, ZTIME : PACKED ARRAY[1..11] OF CHAR;
LASTTOK : BOOLEAN;
BUFFER : ALFA;

 (* SYMBOL TABLE VARIABLES *)

SYMTABLE : ARRAY[1..TABSIZE] OF SYMPTR;
AMULT : REAL;
STRINGHEAD : ARRAY[1..MAXSTRINGS] OF INTEGER;
STRINGSTORE : PACKED ARRAY[1..MAXSSTORE] OF CHAR;

  (* SEMANTIC VARIABLES *)

SWITCH : ARRAY[SWITCHES] OF BOOLEAN;
TEMPLIST : ARRAY[1..TEMPLISTMAX] OF SYMPTR;
TLI : INTEGER;
LINECOUNT,
ALINECOUNT : INTEGER;
FIRSTPARAM : SYMPTR;
LABELCOUNT : INTEGER;


TEMPNAME : ARRAY[1..MAXEVALSTACK] OF RECORD
             NAME : ALFA;
             PRECISION : INTEGER;
             INUSE,USED : BOOLEAN;
            END;

EVALSTACK : ARRAY[1..MAXEVALSTACK] OF RECORD
              NAME : ALFA;
              PRECISION : INTEGER;
              END;


ESI : INTEGER;


CONSTANTSTORE : ARRAY[1..CSSMAX] OF RECORD
                  VAL : INTEGER;
```

```
                        PRECISION : INTEGER;
                        NAME : ALFA;
                       END;
     CSI : INTEGER;


     OPSTORE : ARRAY[1..MAXOPS] OF ALFA;
     OPI : INTEGER;


     SELSTORE : ARRAY[1..MAXSELS] OF INTEGER;
     SELI : INTEGER;
     NLI : INTEGER;

     DESCRIPTION : DESCRIPTOR;


     (* INITIALIZED VALUES GENERATED BY THE AUTOMATIC PARSER *)

    VALUE

     (*LANGUAGE TERMINALS*)
     STRINGSTORE := ('(',')','*','*','*','*','I','D','*','*','N','U','M',
      'B','E','R','*','*','S','T','R','I','N','G','*','+',',','-','.','/',
      '/','=',':',':','=',';','<','<','=','=','=','=','=','>','>','>','=',
      'A','N','D','A','R','I','T','H','M','E','T','I','C','A','S','C','I',
      'I','6','A','S','C','I','I','7','A','T','B','C','D','B','I','N','A',
      'R','Y','C','O','D','E','C','O','N','T','I','N','G','E','N','C','Y',
      'C','O','S','T','C','R','I','T','E','R','I','A','D','A','T','E','D',
      'E','S','I','G','N','D','E','S','I','G','N','E','R','D','O','D','U',
      'P','L','E','X','E','B','C','I','D','I','C','E','C','L','E','N','D',
      'E','N','V','I','R','O','N','M','E','N','T','E','V','E','R','Y','F',
      'I','R','S','T','F','O','R','F','R','O','M','F','U','N','C','T','I',
      'O','N','H','I','D','E','N','T','I','F','I','C','A','T','I','O','N',
      'I','F','I','T','L','I','N','I','N','P','U','T','I','S','S','U','E',
      'L','I','S','T','M','M','E','T','R','I','C','M','O','N','I','T','O',
      'R','S','M','S','N','O','T','N','S','O','R','O','U','T','P','U','T',
      'P','O','W','E','R','P','R','O','C','E','D','U','R','E','S','P','R',
      'O','J','E','C','T','S','S','E','N','S','E','T','A','S','K','T','E',
      'R','M','T','H','E','N','T','O','T','T','L','U','N','T','I','L','U',
      'S','V','A','R','I','A','B','L','E','S','V','O','L','U','M','E','S',
      'W','A','I','T','W','H','E','N','W','H','I','L','E','(',')',
       666 OF ' ');




     (*IN STRINGSTORE THE POSITION OF THE BEGINNING OF EACH TERMINAL*)
     STRINGHEAD := (1,2,3,4,6,10,18,26,27,28,29,30,31,
      33,34,36,37,38,40,41,43,45,46,48,51,61,67,73,75,78,
      84,88,99,103,111,115,121,129,131,137,144,147,150,161,166,171,174,
      178,186,187,201,203,206,208,213,218,222,223,229,237,239,242,244,246,
      252,257,267,274,275,280,284,288,292,294,297,302,304,313,320,324,328,
                 333,334,335,16 OF 0);


(* THE EMITS FROM THE CONFIGERATION SETS IN TERMS *)
(* OF TERMINAL NUMBER                             *)
ENT := (3,42,50,106,129,37,42,36,112,14,34,43,114,7,58,
 25,30,31,39,54,64,87,96,102,103,109,110,113,133,143,149,66,
 151,35,33,45,65,138,14,14,14,77,5,173,174,14,14,110,16,
 48,70,126,127,150,152,165,166,32,105,14,16,5,85,86,5,93,
 94,5,14,9,42,174,16,173,173,16,5,5,16,152,16,16,56,
 7,78,1,162,42,86,16,162,42,94,16,9,5,99,100,6,95,
 39,16,42,42,1,125,125,148,149,182,16,182,51,89,104,116,136,
 153,158,178,67,6,142,142,9,25,16,9,30,16,95,14,42,100,
 16,9,54,64,5,128,14,149,16,5,46,51,53,55,69,79,81,
 88,92,107,108,123,124,130,131,134,135,140,144,160,161,171,172,175,
 176,177,179,180,181,160,1,5,6,7,8,10,61,84,118,119,120,
 121,122,140,147,155,159,168,16,10,28,38,44,80,14,9,16,2,
 6,111,95,16,5,26,27,29,40,101,31,16,41,52,74,167,9,
 14,25,30,16,1,14,82,5,118,6,145,1,1,6,75,118,145,
 38,72,118,140,15,42,161,16,38,118,38,118,42,118,1,71,168,
 20,21,63,24,4,13,17,18,19,22,23,84,156,3,12,139,16,
 145,170,140,164,145,157,140,7,6,59,9,132,132,5,97,98,77,
 5,128,9,9,117,118,5,92,6,47,49,57,60,62,68,76,169,
 140,140,14,160,160,15,5,16,160,14,160,14,5,2,117,119,120,
 121,155,147,168,159,122,38,145,6,72,141,154,38,14,115,142,6,
 14,42,98,16,2,111,95,2,9,14,14,118,2,2,145,42,42,
 42,145,42,6,137,164,140,11,164,157,38,16,6,31,16,132,132,
 118,128,6,73,46,51,53,81,6,146,9,164,2,83,118,90,145,
 14,9,137,91,145,9,154);


(* THE POSITION OF EACH REDUCTION FOR EACH CONFIGURATION SET *)
FRED := (1,1,2,2,2,3,3,4,4,5,5,5,5,6,
 6,6,6,6,6,6,6,6,7,8,9,10,11,11,12,13,14,
 15,15,16,16,17,18,19,19,19,19,19,19,19,19,19,19,19,
 19,20,20,20,21,21,22,22,23,23,23,24,24,24,25,25,25,
 26,26,26,26,26,26,26,26,27,27,27,28,29,30,31,31,32,
 33,34,34,34,34,34,34,34,35,35,35,35,36,36,36,36,36,
 37,37,38,39,39,39,40,40,41,42,42,42,43,43,43,44,44,
 45,47,47,48,49,49,50,50,50,50,51,52,52,53,54,54,54,
 54,54,55,55,56,57,58,58,58,58,59,61,61,61,61,61,61,
 61,62,63,64,65,66,66,67,67,68,69,69,69,70,70,70,71,
 71,72,73,73,74,74,74,74,74,75,76,77,78,79,79,79,80,
 81,82,83,84,85,86,87,88,89,90,90,90,90,90,90,90,90,
 90,91,92,93,94,94,95,96,97,98,99,100,100,101,102,103,104,
 105,105,106,106,106,107,108,108,108,108,109,109,110,110,110,111,112,
 112,113,113,113,114,115,116,116,116,117,117,117,117,117,117,117,118,
 119,120,120,120,120,120,120,121,122,123,124,125,126,126,126,127,128,
 128,129,130,130,131,131,132,132,133,134,135,135,135,136,137,137,137,
 137,138,139,139,139,139,139,140,142,143,143,143,144,145,146,147,148,
 149,150,150,150,150,150,150,151,152,153,153,153,153,153,154,155,155,
 156,157,158,159,160,161,162,163,163,164,165,165,166,167,167,167,167,
 167,168,168,168,168,169,170,171,172,173,173,174,174,174,175,176,177,
 177,177,177,178,178,179,180,181,182,182,183,184,184,185,186,187,188,
 189,190,191,191,191,191,192,193,194,195,196,197,197,198,199,200,200,
 201,202,202,202,203,204,205,205,206);


(* THE POSITION OF EACH TRANSITION FOR EACH CONFIGURATION SET *)

FTRN := (1,2,5,6,7,9,10,10,11,13,14,
 15,31,33,34,38,39,40,42,45,46,47,47,47,
 47,47,62,63,63,63,63,63,71,73,74,74,74,
 74,75,78,81,82,83,84,87,88,91,94,95,95,
 96,97,97,98,105,106,106,107,108,108,109,110,112,
 115,116,118,121,122,123,126,128,129,130,130,133,136,
 136,138,140,150,151,151,161,169,170,172,174,175,176,
 177,177,178,179,180,180,182,183,186,187,187,188,188,
 188,189,190,192,193,193,201,202,232,232,262,280,280,
 281,281,288,292,292,292,293,293,295,297,299,299,299,
 301,301,301,302,308,309,310,310,314,314,314,314,316,
 318,319,319,322,323,341,343,344,345,365,365,365,365,
 365,365,366,366,367,367,367,385,386,386,416,417,417,
 418,418,418,436,436,437,455,485,503,505,505,505,505,
 505,506,514,515,516,517,518,519,519,519,519,529,532,
 532,533,536,539,542,544,545,546,547,547,547,549,551,
 554,554,554,554,554,554,554,555,555,555,555,555,555,
 556,558,559,560,560,579,606,607,608,609,616,616,618,
 620,627,627,629,629,659,689,690,691,691,692,693,693,
 723,725,755,757,758,760,779,779,782,799,815,830,844,
 850,850,850,850,850,850,850,858,871,871,871,878,878,
 878,881,881,885,885,886,888,888,888,890,891,891,891,
 892,895,896,896,896,898,900,902,905,906,908,908,909,
 927,927,927,927,927,927,927,927,928,929,931,961,991,
 991,991,991,1021,1023,1053,1055,1055,1055,1057,1058,1059,1060,
 1060,1060,1063,1066,1067,1070,1070,1070,1072,1073,1073,1076,1079,
 1080,1082,1082,1083,1084,1085,1085,1085,1087,1089,1089,1107,1109,
 1110,1112,1112,1112,1112,1113,1114,1115,1115,1116,1116,1116,1117,
 1117,1119,1120,1121,1124,1124,1124,1124,1124,1124,1124,1125,1127,
 1128,1146,1146,1146,1146,1146,1146,1146,1149,1150,1150,1150,1152,
 1153,1153,1155,1158,1158,1159,1159,1162,1162);


(*  TRANSITIONS *)


TRAN := ( 2,3,4,5,6,7,8,9,10,11,12,13,14,15,
 16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,
 33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,16,17,
 18,19,20,21,22,23,24,25,48,28,29,30,31,49,50,51,52,
 53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,
 70,43,71,72,73,43,74,45,43,75,45,76,77,78,79,50,51,
 52,53,80,56,57,81,82,83,84,85,86,87,62,88,89,90,86,
 91,65,92,93,94,95,96,97,98,99,100,101,102,43,103,72,43,
 104,72,105,106,105,107,16,17,18,22,23,24,25,108,109,110,111,
 16,17,18,22,23,24,25,108,109,112,113,114,115,116,117,118,119,
0372          120,121,122,123,122,124,125,126,127,128,129,130,99,131,132,96,133,
0373          134,135,136,137,138,139,140,141,16,17,18,22,23,24,25,142,143,
0374          144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,
0375          161,162,163,164,165,166,167,168,169,170,171,172,173,144,145,146,147,
0376          148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,174,
0377          165,166,167,168,169,170,171,172,173,175,176,177,178,179,180,181,182,
0378          183,184,185,186,187,188,189,190,191,192,193,113,114,194,116,118,119,
0379          120,195,196,197,198,199,200,201,202,200,203,204,99,205,206,207,208,
0380          209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,
0381          226,227,175,176,177,178,179,180,181,182,228,184,185,186,187,188,189,
0382          190,191,192,229,230,231,232,175,176,233,178,179,180,181,234,182,235,
0383          184,185,186,187,188,236,189,190,191,192,237,238,175,176,177,178,179,
0384          180,181,182,239,184,185,186,187,240,189,190,191,192,241,144,242,145,
0385          146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,
0386          163,243,166,167,168,169,170,171,172,173,244,245,175,176,177,178,179,
0387          180,181,182,246,184,185,186,187,188,189,190,191,192,247,175,176,177,
0388          178,179,180,181,182,248,184,185,186,187,188,189,190,191,192,144,249,
0389          145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,
0390          162,163,243,166,167,168,169,170,171,172,173,175,176,177,178,179,180,
0391          181,182,250,184,185,186,187,188,189,190,191,192,251,226,252,175,176,
0392          177,178,187,188,189,253,254,255,256,257,258,179,180,259,260,261,262,
0393          263,264,265,266,267,268,269,270,229,271,272,176,273,274,229,275,276,
0394          176,277,278,279,280,281,282,281,283,284,285,286,287,288,139,289,290,
0395          291,175,176,177,178,179,180,181,182,292,293,184,185,186,187,188,189,
0396          190,191,192,294,145,146,147,148,149,150,151,152,295,154,155,156,157,
0397          158,159,161,162,163,166,167,168,169,170,171,172,173,296,297,254,298,
0398          299,300,301,302,303,304,176,305,176,306,298,299,300,301,302,303,304,
0399          307,254,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,
0400          159,160,161,162,163,308,165,166,167,168,169,170,171,172,173,144,145,
0401          146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,
0402          163,309,165,166,167,168,169,170,171,172,173,254,310,311,312,144,145,
0403          146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,
0404          163,313,165,166,167,168,169,170,171,172,173,314,254,144,145,146,147,
0405          148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,315,
0406          165,166,167,168,169,170,171,172,173,316,254,317,318,254,175,176,177,
0407          178,179,180,181,182,319,293,184,185,186,187,188,189,190,191,192,267,
0408          268,269,175,176,177,178,179,180,181,182,320,185,186,187,188,189,190,
0409          191,192,175,176,177,178,179,180,181,182,321,186,187,188,189,190,191,
0410          192,175,176,177,178,179,180,181,182,322,187,188,189,190,191,192,175,
0411          176,177,178,179,180,181,182,187,188,189,323,191,192,175,176,177,178,
0412          188,324,175,176,177,178,187,188,189,325,175,176,177,178,179,180,181,
0413          182,187,188,189,326,192,175,176,177,178,327,188,189,229,328,329,330,
0414          331,332,333,334,335,336,122,337,338,339,284,340,341,342,343,219,203,
0415          344,99,345,346,347,348,254,224,226,349,175,176,177,178,179,180,181,
0416          182,350,184,185,186,187,188,189,190,191,192,351,352,229,353,144,354,
0417          145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,
0418          162,163,243,166,167,168,169,170,171,172,173,144,355,145,146,147,148,
0419          149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,243,166,
0420          167,168,169,170,171,172,173,144,356,145,146,147,148,149,150,151,152,
0421          153,154,155,156,157,158,159,160,161,162,163,243,166,167,168,169,170,
0422          171,172,173,229,357,144,358,145,146,147,148,149,150,151,152,153,154,
0423          155,156,157,158,159,160,161,162,163,243,166,167,168,169,170,171,172,
0424          173,359,360,346,347,255,256,257,267,268,269,179,180,265,258,176,273,
               361,176,362,363,176,273,364,229,275,365,366,200,367,368,369,370,281,
               371,281,372,175,176,177,178,179,180,181,182,373,184,185,186,187,188,
               189,190,191,192,139,374,375,254,376,377,378,379,380,331,381,382,331,
               383,176,273,384,254,385,219,386,175,176,177,178,179,180,181,182,387,
               184,185,186,187,188,189,190,191,192,229,388,389,331,390,254,391,359,
               392,229,393,394,395,330,332,396);




              NSET := (59,58,61,57,56,51,51,51,51,57,51,51,51,51,
               61,51,51,51,52,51,56,51,61,44,44,35,52,48,51,34,55,
               34,53,35,35,43,51,35,42,51,34,55,51,51,53,61,51,51,
               28,51,35,51,35,35,51,51,19,33,51,49,2,51,51,51,51,
               51,51,51,51,51,51,51,51,9,10,10,1,1,53,3,7,6,
               4,10,10,5,5,8,54,44,43,51,51,51,51,51,51,51,51,
               35,51,51,51,51,29,33,19,62,60,31,2,51,51,30,2,32,
               29,8,8,2,2,2,2,2,2,1,1,54,40,37,41,60,59,
               28,51,51,51,19,19,49,51,51,36,36,36,36,36,36,36,2,
               51,32,51,10,3,7,6,5,10,8,5,4,40,45,50,51,51,
               35,47,51,51,9,29,51,51,51,51,60,60,51,37,51,60,58,
               51,51,35,51,51,19,51,51,51,51,50,50,51,51,9,60,41,
               60,60,41,60);



              (* REDUCTIONS *)


              PROD := (188,181,1,186,151,83,82,84,85,187,91,89,90,88,
               179,184,183,185,92,139,152,138,190,106,106,97,93,136,136,141,149,
               141,160,103,100,110,94,98,130,145,142,150,176,175,160,180,174,173,
               108,96,104,95,101,118,86,87,131,143,44,133,41,71,78,72,73,
               69,67,79,70,74,76,75,68,133,12,13,2,3,161,32,30,28,
               18,14,16,26,24,20,177,107,111,112,112,125,126,127,129,128,120,
               119,115,116,114,99,130,144,34,38,62,12,66,63,49,14,47,80,
               34,22,21,11,6,7,8,9,10,4,5,178,59,171,153,162,189,
               109,105,102,117,132,35,133,44,77,52,53,55,57,54,56,58,48,
               146,81,140,15,33,31,29,27,17,23,25,19,60,158,156,168,113,
               122,137,112,112,134,130,51,50,64,65,46,40,170,172,169,163,182,
               124,121,123,148,147,36,42,37,61,39,159,157,167,45,135,164,154,
               43,165,155,166);


              (* LOOK AHEAD SETS *)
              LS := (1,5,6,7,1,5,6,7,8,10,61,2,9,14,16,20,28,38,44,72,
               73,80,2,3,8,9,10,12,13,14,16,17,18,19,20,21,22,23,24,28,38,44,
               63,72,73,80,2,9,14,16,20,21,24,28,38,44,63,72,73,80,2,9,14,16,
               20,21,28,38,44,63,72,73,80,2,9,14,16,20,21,28,38,44,72,73,80,2,
               8,9,10,13,14,16,17,18,19,20,21,22,23,24,28,38,44,63,72,73,80,2,
               3,4,6,8,9,10,12,13,14,15,16,17,18,19,20,21,22,23,24,28,38,44,
               63,72,73,80,2,3,4,8,9,10,12,13,14,16,17,18,19,20,21,22,23,24,
               28,38,44,63,72,73,80,2,9,14,16,20,28,38,44,72,80,2,3,8,9,10,
 12,13,14,16,17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,9,14,16,
 20,21,24,28,38,44,63,72,80,2,9,14,16,20,21,28,38,44,63,72,80,2,
 9,14,16,20,21,28,38,44,72,80,2,8,9,10,13,14,16,17,18,19,20,21,
 22,23,24,28,38,44,63,72,80,2,3,4,6,8,9,10,12,13,14,15,16,17,
 18,19,20,21,22,23,24,28,38,44,63,72,80,2,3,4,8,9,10,12,13,14,
 16,17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,9,14,2,3,8,10,
 12,13,14,16,17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,14,16,20,
 21,24,28,38,44,63,72,80,2,14,16,20,21,28,38,44,63,72,80,2,14,16,
 20,28,38,44,72,80,2,14,16,20,21,28,38,44,72,80,2,8,10,13,14,16,
 17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,3,4,8,10,12,13,14,
 15,16,17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,3,4,8,10,12,
 13,14,16,17,18,19,20,21,22,23,24,28,38,44,63,72,80,2,9,16,2,9,
 3,4,8,10,12,13,16,17,18,19,20,21,22,23,24,63,3,4,8,10,12,13,
 14,17,18,19,20,21,22,23,24,63,5,42,46,51,53,55,69,79,81,5,25,30,
 31,46,51,53,55,69,79,81,5,46,51,53,55,69,79,81,5,42,6,9,16,38,
 6,16,72,6,16,38,6,72,6,38,9,38,9,14,9,16,9,11,16,38,11,16,
 14,16,14,15,16,38,16,25,30,31,32,39,42,54,64,66,28,38,44,80,28,28,
 42,44,51,80,32,42,48,70,32,42,32,42,66,32,42,43,66,32,36,42,43,66,
 38,42,72);


(* START LOCATION IN LS OF EACH LOOK AHEAD SET *)

LSET := (1,5,12,23,47,61,74,86,108,135,160,170,193,
 206,218,229,250,276,300,303,325,337,348,357,367,387,
 411,434,437,439,455,471,480,491,499,501,505,508,511,
 513,515,517,519,521;522,525,527,529,530,531,533,534,
 543,547,553,557,559,562,566,571,572,573,574);




LEN := (3,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,3,1,3,1,2,2,3,1,3,1,3,1,3,
 1,3,1,3,0,1,3,5,2,5,4,1,5,8,1,6,1,2,3,2,4,4,1,1,1,1,1,1,2,1,2,5,2,
 2,4,4,2,1,1,1,1,1,1,1,1,1,1,3,1,1,2,3,1,1,1,1,5,5,1,1,1,1,2,3,4,5,
 5,2,3,5,2,3,5,2,3,5,0,3,1,3,1,1,0,2,1,1,1,7,2,3,3,9,2,3,3,1,1,1,1,
 1,0,1,3,1,4,6,0,5,1,1,6,0,1,2,3,3,6,8,8,2,3,0,2,1,1,1,1,3,1,1,0,2,
 0,2,4,6,8,6,4,5,5,1,3,1,1,1,1,2,3,0,3,0,11,1,1,1,0,2,0,10,5);



(* LEFT HAND SIDE OF EACH CONFIGURATION SET PRODUCTION RULE
   BY PRODUCTION NUMBER *)

LHS := (163,84,84,139,139,156,156,156,156,156,156,147,147,147,147,
 122,122,168,168,159,159,159,159,155,155,121,121,120,120,119,119,118,
 118,117,117,117,131,130,179,180,181,124,123,144,144,137,135,135,88,
 107,108,169,169,169,169,169,169,145,170,170,171,172,176,176,175,177,
 92,92,92,92,92,92,92,92,92,92,134,161,161,160,160,149,149,
 149,149,133,143,110,110,110,110,109,109,113,96,87,173,173,174,93,
 93,94,85,85,86,162,162,142,142,95,111,132,132,167,167,167,103,
 99,99,100,102,97,97,98,101,101,101,101,101,128,128,128,140,140,
 140,125,125,152,152,165,182,182,148,148,166,126,127,127,150,150,151,
     151,157,90,91,154,154,141,146,153,153,115,115,115,115,115,178,158,
     116,89,164,164,104,104,104,104,136,136,105,105,112,112,138,138,138,
     114,114,129,129,106);
            .

   (* PARSER INITIALIZATION *)

  FUNCTION HASH (SYM :ALFA) : INTEGER;
   (* USED IN ENTER TO CREATE THE INDEX IN THE SYMBOL *)
   (* TABLE FOR EACH SYMBOL                           *)

    VAR  KEY : INTEGER;
   BEGIN
    KEY := 17*ORD(SYM[1]) + 15*ORD(SYM[2]) + 13*ORD(SYM[3]) +
            3*ORD(SYM[5]) + ORD(SYM[7]);
    HASH := ROUND(KEY*AMULT)MOD TABSIZE + 1
   END; (*HASH*)


  FUNCTION ENTER (SYM : ALFA) : SYMPTR;
   (* ENTERS VALUES IN THE SYMBOL TABLE AND *)
   (* RETURNS THE POINTER TO THE SYMBOL     *)

    VAR PTR : SYMPTR;
        HASHINDEX : INTEGER;
   BEGIN
    NEW(PTR);
    PTR.SYMNAME := SYM;
    PTR.KIND := UNDEFINED;
    HASHINDEX := HASH(SYM);
    PTR.LINK := SYMTABLE[HASHINDEX];
    SYMTABLE[HASHINDEX] := PTR;
    ENTER := PTR
   END; (*ENTER*)


  FUNCTION LOOKUP (SYM : ALFA) : SYMPTR;
   (* LOOKS UP SYMBOLS IN THE SYMBOL TABLE *)
   (* AND RETURNS THE POINTER TO THE SYMBOL DESIRED *)

   VAR PTR, SYMLOC : SYMPTR;
       HASHINDEX : INTEGER;

   BEGIN
    SYMLOC := NIL;
    HASHINDEX := HASH(SYM);
    PTR := SYMTABLE[HASHINDEX];
    WHILE PTR <> NIL DO
     IF SYM = PTR.SYMNAME THEN
      BEGIN
       SYMLOC := PTR;
       PTR := NIL
      END
   ELSE
    PTR := PTR.LINK;
  LOOKUP := SYMLOC
 END; (*LOOKUP*)


PROCEDURE HEADER;
 (* PUTS HEADER IN TRANSLATE FILE *)

 BEGIN
  PAGENUMBER := PAGENUMBER + 1;
  WRITELN (TRANSLATE,'CSDL TRANSLATOR  ',' ':50,'PAGE ',PAGENUMBER : 1);
  WRITELN (TRANSLATE,'NAVAL POSTGRADUATE SCHOOL',' ':47);
  WRITELN (TRANSLATE,ZDATE : 10, ZTIME : 10);
  WRITELN(TRANSLATE);
  PAGELINECOUNT := 3
 END; (*HEADER*)


FUNCTION CHARVAL (NUMBER : INTEGER) : ALFA;
 (* USED TO CREATE LOCATION AND TEMPORARY PRIMITIVE VALUES *)

 BEGIN
  CASE NUMBER OF
   0 : CHARVAL := '00';
   1 : CHARVAL := '01';
   2 : CHARVAL := '02';
   3 : CHARVAL := '03';
   4 : CHARVAL := '04';
   5 : CHARVAL := '05';
   6 : CHARVAL := '06';
   7 : CHARVAL := '07';
   8 : CHARVAL := '08';
   9 : CHARVAL := '09';
  10 : CHARVAL := '10';
  11 : CHARVAL := '11';
  12 : CHARVAL := '12';
  13 : CHARVAL := '13';
  14 : CHARVAL := '14';
  15 : CHARVAL := '15';
  16 : CHARVAL := '16';
  17 : CHARVAL := '17';
  18 : CHARVAL := '18';
  19 : CHARVAL := '19';
  20 : CHARVAL := '20';
  END
 END; (*CHARVAL*)


PROCEDURE INITIALIZE;
 (* INITIALIZES ALL VARIABLES IN THE PROGRAM AND *)
 (* CREATES THE STACKS FOR USE IN THE PARSER     *)

 VAR I,J,LEN,IDXCPY : INTEGER;
     K : SWITCHES;
     PTR : SYMPTR;
BEGIN
 REWRITE(PRIMFILE);
 OPEN(TRANSLATE,RECORDLENGTH := 600);
 REWRITE(TRANSLATE);
 OPEN(SYMFILE,RECORDLENGTH := 200);
 REWRITE(SYMFILE);
 RESET(DAT);
 CH := ' ';
 CC := 0;
 LL := 0;
 LASTTOK := FALSE;
 SOURCELINECOUNT := 0;
 PAGENUMBER := 0;
 DATE(ZDATE);
 TIME(ZTIME);
 HEADER;


 SPS['('] :=  1;  SPS[')'] :=  2;  SPS['*'] :=  3;
 SPS['+'] :=  8;  SPS[','] :=  9;  SPS['-'] := 10;
 SPS['.'] := 11;  SPS['/'] := 12;  SPS[':'] := 14;
 SPS[';'] := 16;  SPS['<'] := 17;  SPS['='] := 19;
 SPS['>'] := 22;  SPS['{'] := 82;  SPS['}'] := 83;


 FOR I := 1 TO TABSIZE DO SYMTABLE[I] :=NIL;
 AMULT := 160795.0/262144.0*TABSIZE;


 FOR I := FIRSTRESWD TO LASTRESWD DO
  BEGIN
   LEN := STRINGHEAD[I + 1] - STRINGHEAD[I];
   J := 1;
   BUFFER := STRINGSTORE[STRINGHEAD[I]];
   WHILE J <= LEN -1 DO
    BEGIN
     J := J + 1;
     IF J <= 10 THEN
      BUFFER := BUFFER + STRINGSTORE[STRINGHEAD[I] + (J - 1)];
    END;
   WHILE J < 10 DO
    BEGIN
     J := J + 1;
     BUFFER := BUFFER + ' '
    END;
   PTR := ENTER (BUFFER);
   PTR.KIND := RESWD;
   PTR.KEY1 := I
  END;
  PROGRAMERRFLAG := FALSE;
```

```
      ERRLIST := [ ];
      LINERRPTR := 0;
      OVERFLOWLOGGED := FALSE;
      MAXLINERRORS := LINERRARRAYSIZE - 2;
      NOWSTA := 1;
      NEXTSYM := ENDTOK;
      STKPTR := 1;
      STACK[1].STATE := 1;
      STACK[1].TOK := 0;
      DESCRIPTION.SYMNAME := '          ';
      DESCRIPTION.INTVAL := 0;
      DESCRIPTION.CHARVAL := '@';
      DESCRIPTION.REALVAL := 0.0;
      DESCRIPTION.SYMLOC := NIL;
      BUFFER := '          ';

      FOR K := TRACEPARSE TO PRINTTABLE DO
       SWITCH[K] := FALSE;
      TLI := 0;                     (* START SEMANTIC INITIALIZATION *)
      FOR I := 1 TO MAXTEMPS DO
       BEGIN
        TEMPNAME[I].NAME := '@T       ';
        TEMPNAME[I].NAME := TEMPNAME[I].NAME + CHARVAL(I);
        IF I IN [1..TPOOL8SIZE] THEN
         TEMPNAME[I].PRECISION := 8
        ELSE
         TEMPNAME[I].PRECISION := 16;
        TEMPNAME[I].INUSE := FALSE;
        TEMPNAME[I].USED := FALSE
       END;
       ESI := 0;
       FOR I := 1 TO CSSMAX DO
        BEGIN
         CONSTANTSTORE[I].NAME := '@C       ';
         CONSTANTSTORE[I].NAME := CONSTANTSTORE[I].NAME +
                                     CHARVAL(I);
         CONSTANTSTORE[I].VAL := 0;          (* ADDED COMMENT *)
         CONSTANTSTORE[I].PRECISION := 8;    (* ADDED COMMENT *)
        END;
       CSI := 0;
       NLI := 0
      END; (*INITIALIZE*)

     (* PARSING ROUTINES *)

   PROCEDURE PUTT (TITLE : ALFA);
    (*PLACES TITLE IN PRIMITIVE FILE. *)

     BEGIN
      LINECOUNT := LINECOUNT + 1;
      WRITELN(PRIMFILE,'P',LINECOUNT : 4,'t.generated for:',TITLE : 10,
              ' ' : 10,'**********************');
     END; (*PUTT*)
```

```


PROCEDURE PUTS (PRIMNAME : ALFA; OPI, SELI : INTEGER);
  (*LOADS PRIMITIVE FILE*)

  VAR I, J : INTEGER;
  BEGIN
   LINECOUNT := LINECOUNT + 1;
   WRITE (PRIMFILE, 'P', LINECOUNT : 4, 's.', PRIMNAME : 10, '(');
   IF OPI > 0 THEN
    BEGIN
     FOR I := 1 TO OPI - 1 DO
      BEGIN
       FOR J := 1 TO 10 DO
        IF OPSTORE[I][J] <> ' ' THEN
         WRITE (PRIMFILE, OPSTORE[I][J] : 1);
       WRITE (PRIMFILE, ',');
      END;
     FOR J := 1  TO 10 DO
      IF OPSTORE[OPI][J] <> ' ' THEN
       WRITE (PRIMFILE, OPSTORE[OPI][J] : 1);
     WRITE (PRIMFILE, ':');
    END;
   IF SELI > 0 THEN
    BEGIN
     FOR I := 1 TO SELI - 1 DO
      WRITE (PRIMFILE, SELSTORE[I] : 1, ',');
     WRITE (PRIMFILE, SELSTORE[SELI] : 1, ')');
    END
   ELSE
    WRITE (PRIMFILE, ')');
   WRITELN (PRIMFILE)
  END; (* PUTS *)


PROCEDURE PUTSYM (PRIMNAME : ALFA; OPI, SELI : INTEGER);
  (* LOADS THE SYMBOL TABLE *)

  VAR I, J : INTEGER;
  BEGIN
   WRITE(SYMFILE, 'S.', PRIMNAME : 10, '(');
   IF OPI > 0 THEN
    BEGIN
     FOR I := 1 TO OPI - 1 DO
      BEGIN
       FOR J := 1 TO 10 DO
        IF OPSTORE[I][J] <> ' ' THEN
         WRITE(SYMFILE, OPSTORE[I][J] : 1);
       WRITE(SYMFILE, ',');
      END;
     FOR J := 1 TO 10 DO
      IF OPSTORE[OPI][J] <> ' ' THEN
       WRITE(SYMFILE, OPSTORE[OPI][J] : 1);
     WRITE(SYMFILE, ':');
    END;
   IF SELI > 0 THEN
    BEGIN
     FOR I := 1 TO SELI - 1 DO
      WRITE(SYMFILE, SELSTORE[I] : 1, ',');
     WRITE(SYMFILE, SELSTORE[SELI] : 1, ')');
    END
   ELSE
    WRITE(SYMFILE, ')');
   WRITELN(SYMFILE)
  END;  (* PUTSYM *)

PROCEDURE GETSYM (VAR NEXTSYM : INTEGER; VAR DES : DESCRIPTOR); FORWARD;
PROCEDURE ERROR (ERRORNUM, INTENSITY,PTROFFSET : INTEGER); FORWARD;

PROCEDURE PUTA (CONTINAME, TASKNAME : ALFA; SELI : INTEGER);
 (* UNITS (MS),RHO,BETA1,BETA2,ORDER,PI,GAMMA1,GAMMA2,BCKGRD *)
 (* LOADS THE TIMING CONSTRAINTS                             *)

 VAR I : INTEGER;
 BEGIN
  ALINECOUNT := ALINECOUNT + 1;
  WRITE(PRIMFILE, 'A', ALINECOUNT : 4,' :',CONTINAME : 10, ':',
        TASKNAME : 10,':MS:');
  FOR I := 1 TO SELI - 1 DO
   WRITE(PRIMFILE, SELSTORE[I] : 4, ',');
  WRITELN(PRIMFILE,SELSTORE[SELI] : 4)
 END; (* PUTA *)

PROCEDURE PUTD (METRIC : ALFA; NUMVOLUMES,NUMONITORS : INTEGER);
 (* LOADS THE DESIGN CRITERIA *)

 VAR I : INTEGER;
 BEGIN
  LINECOUNT := LINECOUNT + 1;
  WRITE(PRIMFILE, 'P',LINECOUNT : 4,'d:', METRIC : 10, ':');
  FOR I := 1 TO NUMVOLUMES - 1 DO
   WRITE(PRIMFILE, SELSTORE[I] : 1, ',');
  WRITE(PRIMFILE, SELSTORE[NUMVOLUMES], ':');
  FOR I := NUMVOLUMES + 1 TO NUMVOLUMES + NUMONITORS DO
   WRITE(PRIMFILE, SELSTORE[I], ',');
  WRITELN(PRIMFILE, SELSTORE[NUMVOLUMES + 1 + NUMONITORS], ':')
 END; (* PUTD *)

FUNCTION CHECKTYPE (EXP1TYPE, EXP2TYPE : EXPTYPES) : EXPTYPES;
 (* CHECKS THE TYPES OF INPUT VALUES *)

 BEGIN
  IF EXP1TYPE = EXP2TYPE THEN
   CHECKTYPE := EXP1TYPE
  ELSE
   CHECKTYPE := ERRORS
 END; (* CHECKTYPE *)

FUNCTION COMPUTPRE (PRE1, PRE2 : INTEGER) : INTEGER;
 (* COMPUTS THE PRECISION REQUIRED FOR THE *)
 (* RESULT OF AN ARITHMETIC OPERATION      *)

 BEGIN
  IF PRE1 > PRE2 THEN
   COMPUTPRE := PRE1
  ELSE
   COMPUTPRE := PRE2
 END; (* COMPUTPRE *)

PROCEDURE NEWTEMP (VAR ZPRECISION : INTEGER; VAR TEMP : ALFA);
 (* CHECKS TO SEE IF A VARIABLE NAME HAS BEEN PREVIOUSLY *)
 (* DEFINED. IF IT HAS, NEWTEMP RETURNS TRUE. IF NOT, IT *)
 (* RETURNS FALSE.                                       *)

 CONST LAST8 = 10;
 VAR I, STOPSEARCH : INTEGER;
 BEGIN
  CASE ZPRECISION OF
   1,2,3,4,5,6,7,8 :
    BEGIN
     I := 1;
     STOPSEARCH := LAST8
    END;
   9,10,11,12,13,14,15,16 :
    BEGIN
     I := LAST8 + 1;
     STOPSEARCH := MAXTEMPS
    END;
   OTHERWISE ERROR(35,1,-1);
  END;
  WHILE (I <= STOPSEARCH) AND (TEMPNAME[I].INUSE) DO
   I := I + 1;
  IF I > STOPSEARCH THEN
   ERROR(22,1,-1)
  ELSE
   BEGIN
    TEMP := TEMPNAME[I].NAME;
    ZPRECISION := TEMPNAME[I].PRECISION;
    TEMPNAME[I].INUSE := TRUE;
    IF NOT TEMPNAME[I].USED THEN
     TEMPNAME[I].USED := TRUE
   END
 END; (* NEWTEMP *)

PROCEDURE RETURNTEMP (TEMP : ALFA);
 (* RETURNS FALSE IF A TEMP IS NEW *)

 VAR I : INTEGER;
 BEGIN
```

```
  I := 1;
  WHILE (TEMPNAME[I].NAME <> TEMP) AND (I <= MAXTEMPS) DO
   I := I + 1;
  IF I > MAXTEMPS THEN
   WRITELN(TRANSLATE, ' FIX THE ERROR IN RETURNTEMP')
  ELSE
   TEMPNAME[I].INUSE := FALSE
 END; (* RETURNTEMP *)



PROCEDURE PRINTEMPS;
 (* LOADS THE VARIABLES *)

 VAR I : INTEGER;
 BEGIN
  FOR I := 1 TO MAXTEMPS DO
   IF TEMPNAME[I].USED THEN
    BEGIN
     OPSTORE[1] := TEMPNAME[I].NAME;
     SELSTORE[1] := TEMPNAME[I].PRECISION;
     PUTS('var        ', 1,1)
    END
 END; (* PRINTEMPS *)


PROCEDURE PUSHEVALSTACK (ZNAME : ALFA; ZPRECISION : INTEGER);
 (* PUSHES A VARIABLE ON TO THE STACK *)

 BEGIN
  ESI := ESI + 1;
  EVALSTACK[ESI].NAME := ZNAME;
  EVALSTACK[ESI].PRECISION := ZPRECISION
 END; (*PUSHEVALSTACK *)

PROCEDURE POPEVALSTACK (VAR NAME : ALFA; VAR ZPRECISION : INTEGER);
 (* POPS A VARIABLE FROM THE STACK *)

 BEGIN
  NAME := EVALSTACK[ESI].NAME;
  ZPRECISION := EVALSTACK[ESI].PRECISION;
  IF (NAME[1] = '●') AND (NAME[2] = 'T') THEN
   RETURNTEMP(NAME);
  ESI := ESI - 1
 END; (* POPEVALSTACK *)

PROCEDURE NEWCONS (ZVALUE : INTEGER; VAR ZNAME : ALFA;
                   VAR ZPRECISION : INTEGER);
 (* CREATES A NEW CONSTANT IF NEEDED. OTHERWISE *)
 (* RETURNS THE POINTER TO THE ONE DESIRED      *)

 VAR I : INTEGER;
 BEGIN
```

```
  I := 1;
  WHILE (I <= CSI) AND (CONSTANTSTORE[I].VAL <> ZVALUE) DO
   I := I + 1;
  IF I > CSI THEN
   BEGIN
    CSI := CSI + 1;
    CONSTANTSTORE[CSI].VAL := ZVALUE;
    IF (ZVALUE >= -127) AND (ZVALUE <= 128) THEN
     ZPRECISION := 8
    ELSE
     IF (ZVALUE >= -32768) AND (ZVALUE <= 32768) THEN
      ZPRECISION := 16
     ELSE
      BEGIN
       ERROR(31,1,-1);
       ZPRECISION := 16
      END;
    CONSTANTSTORE[CSI].PRECISION := ZPRECISION;
    ZNAME := CONSTANTSTORE[I].NAME
   END
  ELSE
   BEGIN
    ZNAME := CONSTANTSTORE[I].NAME;
    ZVALUE := CONSTANTSTORE[I].VAL;
    ZPRECISION := CONSTANTSTORE[I].PRECISION
   END
 END; (* NEWCONS *)

PROCEDURE NEWLABEL (VAR LABELNAME : ALFA);

 VAR PLACE, LCCOPY : INTEGER;
 BEGIN
  LABELNAME := '@       ';
  LABELCOUNT := LABELCOUNT + 1;
  LCCOPY := LABELCOUNT;
  IF LCCOPY > 0 THEN
   LABELNAME := LABELNAME + CHARVAL(LCCOPY);
 END; (*  NEWLABEL *)




FUNCTION FINDTRANSITION (CSTATE,CTOKEN : INTEGER) : INTEGER;
  (* CHECKS TO SEE IF ANY TRANSITIONS EXIST *)

  VAR I : INTEGER;
  BEGIN
   FINDTRANSITION := 0;
   FOR I := FTRN[CSTATE] TO FTRN[CSTATE + 1] - 1 DO
    IF CTOKEN = ENT[TRAN[I]] THEN
     FINDTRANSITION := TRAN[I]
  END; (*FINDTRANSITION*)
```

```
FUNCTION FINDREDUCTION (CSTATE,CTOKEN : INTEGER) : INTEGER;
  (* CHECKS TO SEE IF ANY REDUCTIONS EXIST *)

  VAR I, J : INTEGER;
  BEGIN
   FINDREDUCTION := 0;
   FOR I := FRED[CSTATE] TO FRED[CSTATE + 1] - 1 DO
    FOR J := LSET[NSET[I]] TO LSET[NSET[I] + 1] - 1 DO
     IF CTOKEN = LS[J] THEN
      FINDREDUCTION := PROD[I]
  END; (*FINDREDUCTION*)

PROCEDURE DOTRANSITION (NEWSTA : INTEGER);
   (* GIVEN THE TRANSITION NUMBER, THIS MODULE *)
   (* EXECUTES THE TRANSITION AND RETURNS TO   *)
   (* PARSE TO CONTINUE THE LOOP               *)

  BEGIN
   IF SWITCH[TRACEPARSE] THEN
    WRITELN(TRANSLATE, ' TRANSITION FROM STATE ', NOWSTA :2,
            ' TO STATE ', NEWSTA :2);
   STKPTR := STKPTR + 1;
   IF STKPTR <= MAXSTK THEN
    BEGIN
     STACK[STKPTR].TOK := NEXTSYM;
     STACK[STKPTR].DES := DESCRIPTION;
     STACK[STKPTR].STATE := NEWSTA;
     NOWSTA := NEWSTA;
     GETSYM(NEXTSYM,DESCRIPTION);
     IF SWITCH[TRACETOK] THEN
      WRITELN(TRANSLATE, ' NEXTSYM = ', NEXTSYM : 2);
    END
   ELSE
    BEGIN
     WRITELN(TRANSLATE,'GOING INTO ERROR IN DOTRAN');
     ERROR(8,5,0)
    END
  END; (*DOTRANSITION*)

PROCEDURE SEMANTIC (PRODUCTION : INTEGER); FORWARD;
PROCEDURE SEMANTIC1(PRODUCTION : INTEGER); FORWARD;

PROCEDURE DOREDUCTION (PROD : INTEGER);

  (* GIVEN THE REDUCTION, THIS MODULE DOES THE REDUCTION *)
  (* BY CALLING SEMANTIC WITH THE PARTICULAR CONSTRUCT   *)
  (* IT HAS RECOGNIZED. DOREDUCTION THEN REMOVES THE     *)
  (* APPROPRIATE NUMBER OF ITEMS FROM THE STACK AND      *)
  (* PLACES THE REDUCED VERSION ON THE STACK.            *)

  BEGIN
   SEMANTIC(PROD);
   STKPTR := STKPTR - LEN[PROD];
   IF STKPTR <= MAXSTK THEN
    BEGIN
     NOWSTA := FINDTRANSITION(STACK[STKPTR].STATE, LHS[PROD]);
     IF SWITCH[TRACEPARSE] THEN
      WRITELN(TRANSLATE, ' PRODUCTION ', PROD : 2,
              ' AND TRANSITION FROM STATE ', STACK[STKPTR].STATE : 2,
              ' TO STATE ', NOWSTA : 2);
     STKPTR := STKPTR + 1;
     STACK[STKPTR].TOK := LHS[PROD];
     STACK[STKPTR].STATE := NOWSTA
    END
   ELSE
    BEGIN
     WRITELN(TRANSLATE,'GOING INTO ERROR IN DOREDUCTION');
     ERROR(9,5,0)
    END
  END; (* DOREDUCTION *)


PROCEDURE PARSE;

  (* REPEATEDLY EXECUTES UNTIL IT TRANSITIONS TO A FINAL       *)
  (* STATE. CALLS FINDREDUCTION TO SEE IF ANY REDUCTIONS       *)
  (* EXIST. IF ONE DOES, IT DOES IT BY CALLING DOREDUCTION     *)
  (* AND GOES BACK TO REPEAT THE LOOP. IF NO REDUCTIONS        *)
  (* EXIST, IT CALLS FINDTRANSITION TO SEE IF ANY TRANSITIONS *)
  (* EXIST. IF ONE DOES, IT CALLS DOTRANSITION AND THEN        *)
  (* REPEATS THIS LOOP. IF NEITHER EXIST, THEN EITHER AN       *)
  (* ERROR EXISTS OR WE HAVE TRANSITIONED TO A FINAL STATE     *)

 BEGIN
  REPEAT
   REDUCTION := FINDREDUCTION(NOWSTA,NEXTSYM);
   IF REDUCTION > 0 THEN
    DOREDUCTION(REDUCTION)
   ELSE
    BEGIN
     TRANSITION := FINDTRANSITION(NOWSTA,NEXTSYM);
     IF TRANSITION <> FSTATE THEN
      BEGIN
       IF TRANSITION > 0 THEN
        DOTRANSITION(TRANSITION)
       ELSE
        BEGIN
         WRITELN(TRANSLATE,'GOING INTO ERROR IN PARSE');
         ERROR(5,4,-1)
        END
      END
     ELSE
      NOWSTA := TRANSITION
    END
  UNTIL NOWSTA =FSTATE
  END; (* PARSE *)




(* ERROR HANDLING ROUTINES *)

PROCEDURE PRINTERRORS;

  VAR I : INTEGER;
  BEGIN
   IF PAGELINECOUNT + 10 > PAGESIZE THEN
    WRITELN(TRANSLATE,'1');
   I := 1;
   WRITELN(TRANSLATE);
   WRITELN(TRANSLATE);
   WRITELN(TRANSLATE, ' ':25,'PROGRAM ERRORS');
   WRITELN(TRANSLATE,' ':25,'**************');
   WRITELN(TRANSLATE);
   WRITELN(TRANSLATE);
   WHILE ERRLIST <> [ ] DO
    BEGIN
     WHILE NOT (I IN ERRLIST) DO
      I := I + 1;
     WRITE(TRANSLATE,I : 5);
     CASE I OF
      1 : WRITELN(TRANSLATE, ' : DIGIT EXPECTED');
      2 : WRITELN(TRANSLATE, ' : ERROR IN IDENTIFIER');
      3 : WRITELN(TRANSLATE, ' : ERROR IN NUMBER');
      4 : WRITELN(TRANSLATE, ' : ERROR IN EXPONENT');
      5 : WRITELN(TRANSLATE, ' : IMPROPER CONSTRUCTION, EXPECTED',
                 ' SYMBOL LIST FOLLOWS');
      6 : WRITELN(TRANSLATE, ' : BASED INTEGERS ARE DEFINED ONLY FOR',
                 ' BASES 2 THROUGH 16');
      7 : WRITELN(TRANSLATE, ' : CHARACTER NOT DEFINED FOR THIS BASE');
      8 : WRITELN(TRANSLATE, ' : STACK OVERFLOW IN DOTRANSITION. ',
                 'TO CORRECT, INCREASE CONSTANT MAXSTK.');
      9 : WRITELN(TRANSLATE, ' : STACK OVERFLOW IN DOREDUCTION. ',
                 'TO CORRECT, INCREASE CONSTANT MAXSTK.');
      10: WRITELN(TRANSLATE, ' : END OF FILE ENCOUNTERED.');
      11: WRITELN(TRANSLATE, ' : UNKNOWN OR MISPLACED CHARACTER');
      12: WRITELN(TRANSLATE, ' : TOO MANY ERRORS ON THIS LINE');
      13: WRITELN(TRANSLATE, ' : INTEGER VALUE EXPECTED');
      14: WRITELN(TRANSLATE, ' : IDENTIFIER ALREADY DECLARED');
      15: WRITELN(TRANSLATE, ' : CHARACTER ALREADY SPECIFIED');
      16: WRITELN(TRANSLATE, ' : CHARACTER REPRESENTATION MUST BE AN',
                 ' INTEGER VALUE');
      17: WRITELN(TRANSLATE, ' : UNKNOWN OR MISSING COMPILER OPTION',
                 ' INSTRUCTION');
      18: WRITELN(TRANSLATE, ' : REALS NOT IMPLEMENTED');
      19: WRITELN(TRANSLATE, ' : OPERATION NOT DEFINED FOR STRING',
                 'OPERANDS');
```

```
          20: WRITELN(TRANSLATE, ' : TYPE MISMATCH');
          21: WRITELN(TRANSLATE, ' : OPERATION NOT DEFINED FOR BOOLEAN',
                  ' OPERANDS');
          22: WRITELN(TRANSLATE, ' : TOO MANY REQUESTS FOR TEMP NAMES--',
                  ' INCREASE CONSTANT MAXTEMPS');
          23: WRITELN(TRANSLATE, ' : OPERATION NOT DEFINED FOR NUMERICAL',
                  ' OPERANDS');
          24: WRITELN(TRANSLATE, ' : PROCEDURE NAME EXPECTED');
          25: WRITELN(TRANSLATE, ' : MISMATCHING BEGIN-END PAIR');
          26: WRITELN(TRANSLATE, ' : EXPRESSION TYPE MUST BE BOOLEAN');
          27: WRITELN(TRANSLATE, ' : ERROR IN FACTOR');
          28: WRITELN(TRANSLATE, ' : IDENTIFIER NOT DECLARED');
          29: WRITELN(TRANSLATE, ' : INDEX MUST BE DECLARED AS ARITHMETIC');
          30: WRITELN(TRANSLATE, ' : THE FOLLOWING ARGUMENT MUST ',
                  'BE BINARY');
          31: WRITELN(TRANSLATE, ' : CONSTANT TOO LARGE--NOT IN',
                 ' -32768..32768');
          32: WRITELN(TRANSLATE, ' : IMPROPER ASSIGNMENT');
          33: WRITELN(TRANSLATE, ' : PARAMETERS NOT IMPLEMENTED');
          34: WRITELN(TRANSLATE, ' : TASK NAME EXPECTED');
          35: WRITELN(TRANSLATE, ' : REQUEST FOR A TEMPORARY WITH A',
                  ' PRECISION > 16');
          36: WRITELN(TRANSLATE, ' : STRUCTURED VARIABLES ARE ',
                  'NOT IMPLEMENTED');
          37: WRITELN(TRANSLATE, ' : BIT FIELDS ARE NOT IMPLEMENTED');
          38: WRITELN(TRANSLATE, ' : EXPRESSION TYPES MUST BE INTEGER');
        END;
        ERRLIST := ERRLIST - [I]
       END
     END; (*PRINTERRORS*)


   PROCEDURE RECOVER;
    (* THIS ATTEMPTS TO RECOVER THE PARSE FROM AN *)
    (* ERROR SUCH AS AN UNDEFINED VARIABLE        *)

     CONST NUMSOLIDS = 21;
     VAR SOLID : ARRAY[1..NUMSOLIDS] OF INTEGER;

     FUNCTION SOLIDTOKEN (TOKEN : INTEGER) : BOOLEAN;
      VAR LOWERBOUND, UPPERBOUND, INDEX : INTEGER;
          FOUND : BOOLEAN;
      BEGIN
       LOWERBOUND := 1;
       UPPERBOUND := NUMSOLIDS;
       FOUND := FALSE;
       WHILE (LOWERBOUND < UPPERBOUND) AND (NOT FOUND) DO
        BEGIN
         INDEX := (LOWERBOUND + UPPERBOUND) DIV 2;
         IF TOKEN < SOLID[INDEX] THEN
          UPPERBOUND := INDEX - 1
         ELSE
          IF TOKEN > SOLID[INDEX] THEN
        LOWERBOUND := INDEX + 1
       ELSE
        FOUND := TRUE
     END; (*WHILE*)
    SOLIDTOKEN := FOUND
   END; (* SOLIDTOKEN *)

  BEGIN
   SOLID[ 1] :=  85;  SOLID[ 2] :=  92;  SOLID[ 3] :=  97;
   SOLID[ 4] :=  99;  SOLID[ 5] := 105;  SOLID[ 6] := 109;
   SOLID[ 7] := 112;  SOLID[ 8] := 114;  SOLID[ 9] := 117;
   SOLID[10] := 128;  SOLID[11] := 129;  SOLID[12] := 135;
   SOLID[13] := 136;  SOLID[14] := 142;  SOLID[15] := 148;
   SOLID[16] := 150;  SOLID[17] := 151;  SOLID[18] := 160;
   SOLID[19] := 164;  SOLID[20] := 170;  SOLID[21] := 173;

   WHILE (NOT SOLIDTOKEN(STACK[STKPTR].TOK)) AND
         (STKPTR > 2) DO
    STKPTR := STKPTR - 1;
   WHILE (FINDTRANSITION(STACK[STKPTR].STATE,NEXTSYM) = 0) AND
         (FINDREDUCTION(STACK[STKPTR].STATE,NEXTSYM) = 0) AND
         (NOT LASTTOK) DO
    GETSYM(NEXTSYM,DESCRIPTION);
   IF LASTTOK THEN
    ERROR(10,5,-1);
   NOWSTA := STACK[STKPTR].STATE
  END; (*RECOVER *)


PROCEDURE ERROR (*ERRORNUM, INTENSITY, PTROFFSET : INTEGER *);

  VAR POSITION : INTEGER;
  BEGIN
   WRITELN(TRANSLATE,'IN ERROR SLC = ',SOURCELINECOUNT : 3);
   IF NOT PROGRAMERRFLAG THEN
    PROGRAMERRFLAG := TRUE;
   IF LINERRPTR < MAXLINERRORS THEN
    BEGIN
     LINERRPTR := LINERRPTR + 1;
     LINERRORS[LINERRPTR].ERRPOSITION := CC + PTROFFSET;
     LINERRORS[LINERRPTR].ERRNUM := ERRORNUM;
     LINERRORS[LINERRPTR].STATE := NOWSTA;
     ERRLIST := ERRLIST + [ERRORNUM];
    END
   ELSE
    IF NOT OVERFLOWLOGGED THEN
     BEGIN
      OVERFLOWLOGGED := TRUE;
      LINERRPTR := LINERRPTR + 1;
      LINERRORS[LINERRPTR].ERRPOSITION := CC + PTROFFSET;
      LINERRORS[LINERRPTR].ERRNUM := 12;
      ERRLIST := ERRLIST + [12]
     END;
   IF INTENSITY = 5 THEN
    GOTO 99;
   IF INTENSITY = 4 THEN
    RECOVER
  END; (* ERROR *)


PROCEDURE PRINTLINERRORS;

  VAR LINEPOSITION, MARKER : INTEGER;
      MORERRORS : BOOLEAN;

  PROCEDURE PRINTERROR (X : INTEGER);
   VAR I, J, K, SYM : INTEGER;
   BEGIN
    WITH LINERRORS[X] DO
     BEGIN
      IF ERRPOSITION = LINEPOSITION + 1 THEN
       BEGIN
        WRITE(TRANSLATE,'',ERRNUM : 2);
        LINEPOSITION := LINEPOSITION + 3
       END
      ELSE
       BEGIN
        WRITE(TRANSLATE,' ': (ERRPOSITION - LINEPOSITION) - 1,'',
              ERRNUM : 2);
        LINEPOSITION := ERRPOSITION + 2;
       END;
     IF ERRNUM = 5 THEN
      BEGIN
       WRITE(TRANSLATE,' ');
       LINEPOSITION := LINEPOSITION + 1;
       FOR I := FTRN[STATE] TO FTRN[STATE + 1] DO
        BEGIN
         SYM := ENT[TRAN[I]];
         IF SYM <= NUMTERMINALS THEN
          BEGIN
           FOR J := STRINGHEAD[SYM] TO STRINGHEAD[SYM + 1] - 1 DO
            BEGIN
             WRITE(TRANSLATE, STRINGSTORE[J] : 1);
             LINEPOSITION := LINEPOSITION + 1
            END;
           WRITE(TRANSLATE,' ');
           LINEPOSITION := LINEPOSITION + 1
          END
         END;
        FOR I := FRED[STATE] TO FRED[STATE + 1] DO
         FOR J := LSET[NSET[I]] TO LSET[NSET[I] + 1] - 1 DO
          BEGIN
           SYM := LS[J];
           FOR K := STRINGHEAD[SYM] TO STRINGHEAD[SYM + 1] DO
            BEGIN
             WRITE(TRANSLATE,STRINGSTORE[K] : 1);
             LINEPOSITION := LINEPOSITION + 1
            END;
           WRITE(TRANSLATE,' ');
           LINEPOSITION := LINEPOSITION + 1
          END
       END
     END
    END;

  BEGIN
   MORERRORS := TRUE;
   LINEPOSITION := 0;
   MARKER := 1;
   WRITE(TRANSLATE,' **** ');
   WHILE MORERRORS DO
    BEGIN
     WHILE MARKER <= LINERRPTR DO
     BEGIN
     IF LINERRORS[MARKER].ERRPOSITION > LINEPOSITION THEN
      BEGIN
       PRINTERROR(MARKER);
       LINERRORS[MARKER].ERRPOSITION := 0
      END;
     MARKER := MARKER + 1
    END;
   MARKER := 1;
   WHILE (MARKER <= LINERRPTR) AND
         (LINERRORS[MARKER].ERRPOSITION = 0) DO
    MARKER := MARKER + 1;
   IF MARKER <= LINERRPTR THEN
    BEGIN
     WRITELN(TRANSLATE);
     LINEPOSITION := 0;
     WRITE(TRANSLATE,'      ');
     PAGELINECOUNT := PAGELINECOUNT + 1
    END
   ELSE
    BEGIN
     WRITELN(TRANSLATE);
     MORERRORS := FALSE
    END
   END
  END; (* PRINTLINERRORS *)


PROCEDURE GOPRINTTABLE;
 (* THIS IS USED BY A TOGGLE IN THE INPUT IF *)
 (* THE DETAILS OF THE PARSE ARE DESIRED IN  *)
 (* AN OUTPUT FILE CALLED TRANSLATE          *)

  VAR PTR : SYMPTR;
      I, J : INTEGER;

```

```
BEGIN
 FOR I := 1 TO TABSIZE DO
  BEGIN
   PTR := SYMTABLE[I];
   WRITELN(TRANSLATE,I : 5);
   WHILE PTR <> NIL DO
    BEGIN
     WRITE(TRANSLATE,'     ');
     WRITE(TRANSLATE,PTR.SYMNAME : 10, '   ');
     WITH PTR DO
      CASE KIND OF
       UNDEFINED : WRITE(TRANSLATE,'UNDEFINED');
       RESWD : WRITE(TRANSLATE,'RESWD     ', KEY1 : 5);
       BINARY : WRITE(TRANSLATE, 'BINARY    ', PRECISION2 : 5,
                      IVAL2 : 5);
       ARITHMETIC : WRITE(TRANSLATE, 'ARITHMETIC', PRECISION3 : 5,
                          IVAL3 : 5);
       WORD : WRITE(TRANSLATE, 'TEXT      ', CODID4.SYMNAME : 10,
                    STRINGPTR4 : 5);
       CHAREP : WRITE(TRANSLATE, 'CHARREP   ', CODID5.SYMNAME : 10,
                      IVAL5 : 5);
       TRANSDEC : WRITE(TRANSLATE, 'TRANSDEC ', TYPE6.SYMNAME : 10,
                        TECHNOLOGY6.SYMNAME : 10, PRECISION6 : 5);
       TASK : WRITE(TRANSLATE, 'TASK      ');
       FNCTN : WRITE(TRANSLATE, 'FUNCTION  ', TYPE8.SYMNAME : 10,
                     PRECISION8 : 5)
      END;
     WRITELN(TRANSLATE);
     PTR := PTR.LINK
    END;
   WRITELN(TRANSLATE)
  END
 END; (* GOPRINTTABLE *)

(*--------------*** GET INPUT SYMBOLS ***--------------*)

PROCEDURE GETSYM (*VAR NESTSYM : INTEGER; VAR DES : DESCRIPTOR *);
  (* GETS THE NEXT INPUT SYMBOL AND DETERMINES *)
  (* THE TYPE AND DESCRIPTION OF THE TOKEN     *)

 VAR LEN,                 (* LENGTH OF INPUT SYMBOL *)
     I,                   (* INDEX *)
     BASE,                (* BASE FOR BASED INTEGER *)
     NUMDIGITS,           (* NUMBER OF DIGITS IN CURRENT INTEGER *)
     SCALE,               (* EXPONENT ADJUSTMENT *)
     EXPONENT : INTEGER;(* INTEGER VAL OF EXPONENT *)
     SIGN,                (* FLAG INDICATING NEGATIVE EXPONENT *)
     DECIMALFLAG,         (* FLAG INDICATING RADIX PT *)
     BASEDNUMBER,         (* FLAG INDICATING BASE OTHER THAN 10 *)
     ENDSTRING : BOOLEAN; (* FLAG INDICATING END OF STRING *)
     FAC, R : REAL;       (* FOR EXPONENT ADJUSTMENT *)

 FUNCTION FLIP (FLIPEE : BOOLEAN) : BOOLEAN;
```

```
BEGIN
 IF FLIPEE = TRUE THEN
  FLIP := FALSE
 ELSE
  FLIP := TRUE
END; (*FLIP*)

FUNCTION CHARVAL (CH : CHAR) : INTEGER;
BEGIN
 IF CH IN ['A'..'F','0'..'9'] THEN
  CASE CH OF
   '0' : CHARVAL :=  0;
   '1' : CHARVAL :=  1;
   '2' : CHARVAL :=  2;
   '3' : CHARVAL :=  3;
   '4' : CHARVAL :=  4;
   '5' : CHARVAL :=  5;
   '6' : CHARVAL :=  6;
   '7' : CHARVAL :=  7;
   '8' : CHARVAL :=  8;
   '9' : CHARVAL :=  9;
   'A' : CHARVAL := 10;
   'B' : CHARVAL := 11;
   'C' : CHARVAL := 12;
   'D' : CHARVAL := 13;
   'E' : CHARVAL := 14;
   'F' : CHARVAL := 15
  END
 ELSE
  CHARVAL := 16
END; (* CHARVAL*)

PROCEDURE INCHAR (VAR CH : CHAR);
BEGIN
 IF CC = LL THEN
  BEGIN
     IF EOF(DAT) THEN
      BEGIN
       WRITELN(TRANSLATE,'GOING INTO ERROR IN INCHAR');
       ERROR(10,5,0)
      END;
     IF LINERRPTR <> 0 THEN
      BEGIN
       PRINTLINERRORS;
       LINERRPTR := 0;
       OVERFLOWLOGGED := FALSE
      END;
     LL := 0;
     CC := 0;
     IF PAGELINECOUNT MOD PAGESIZE = 0 THEN  (* NEW PAGE *)
      BEGIN
       WRITELN(TRANSLATE,'1');
       HEADER
       END;
      SOURCELINECOUNT := SOURCELINECOUNT + 1;
      WRITE(TRANSLATE, SOURCELINECOUNT : 5, ' ');
      WHILE NOT EOLN(DAT) DO
       BEGIN
        LL := LL + 1;
        READ(DAT,CH);
        WRITE(TRANSLATE,CH);
        LINE[LL] := CH
       END;
      LL := LL + 1;
      LINE[LL] := ' ';
      WRITELN(TRANSLATE);
      READLN(DAT);
      PAGELINECOUNT := PAGELINECOUNT + 1
     END;
    CC := CC + 1;
    CH := LINE[CC];
    IF EOF(DAT) THEN
     IF CC = LL THEN
      LASTTOK := TRUE
   END; (*INCHAR*)

PROCEDURE GETNUMBER (VAR CH : CHAR; BASE : INTEGER;
           DECIMALPART : BOOLEAN; VAR DIGITS, IVALUE : INTEGER;
           VAR RVALUE : REAL);
 BEGIN
  DIGITS := 1;
  IF CHARVAL(CH) >= BASE THEN
   ERROR(7,1,0);
  IF DECIMALPART THEN
   RVALUE := RVALUE * BASE + CHARVAL(CH)
  ELSE
   IVALUE := CHARVAL(CH);
  INCHAR(CH);
  WHILE ((BASE = 10) AND (CH IN ['0'..'9',''])) OR
        ((BASE <> 10) AND (CH IN ['0'..'9','A'..'Z',''])) DO
   BEGIN
    IF CH = '' THEN
     INCHAR(CH);
    IF DECIMALPART THEN
     RVALUE := RVALUE * BASE + CHARVAL(CH)
    ELSE
     IVALUE := IVALUE * BASE + CHARVAL(CH);
    DIGITS := DIGITS + 1;
    IF CHARVAL(CH) >= BASE THEN
     ERROR(7,1,0);
    INCHAR(CH)
   END
 END; (* GETNUMBER *)


PROCEDURE GETEXPONENT (VAR SIGN : BOOLEAN; VAR EXP : INTEGER);
```

```
  VAR NUMDIGITS, INTVAL : INTEGER;
      REALVAL : REAL;
  BEGIN
   SIGN := FALSE;
   INCHAR(CH);
   IF CH = '+' THEN
    INCHAR(CH)
   ELSE
    IF CH = '-' THEN
     BEGIN
      SIGN := TRUE;
      INCHAR(CH)
     END;
   IF CH IN ['0'..'9'] THEN
    BEGIN
     GETNUMBER(CH,10,FALSE,NUMDIGITS,INTVAL,REALVAL);
     EXP := INTVAL
    END
   ELSE
    BEGIN
     ERROR(4,1,0);
     EXP := 0
    END
  END;

BEGIN (* GETSYM *)
 DES.SYMNAME := '          ';
 DES.INTVAL := 0;
 DES.REALVAL := 0.0;
 DES.TMPNAME := '          ';
 DES.CHARVAL := '●';
 DES.SYMLOC := NIL;
 DES.LINEPOS := LINELENGTH;
 BUFFER := '     ';
 IF EOF(DAT) AND LASTTOK THEN
  NEXTSYM := ENDTOK
 ELSE
  WHILE CH = ' ' DO
   INCHAR(CH);
 CASE CH OF
  'A','B','C','D','E','F','G','H','I',
  'J','K','L','M','N','O','P','Q','R',
  'S','T','U','V','W','X','Y','Z' :
   BEGIN
    I := 1;
    BUFFER[I] := CH;
    INCHAR(CH);
    WHILE CH IN ['A'..'Z','0'..'9',''] DO
     BEGIN
      IF I < 10 THEN
       BEGIN
        I := I + 1;
        BUFFER[I] := CH
     END;
          IF CH = '' THEN
     BEGIN
      INCHAR(CH);
      IF I < 10 THEN
       BEGIN
        I := I + 1;
        BUFFER[I] := CH
       END;
      IF NOT (CH IN ['A'..'Z','0'..'9']) THEN
       ERROR(2,1,0)
     END;
    INCHAR(CH)
   END;
  WHILE I < 10 DO
   BEGIN
    I := I + 1;
    BUFFER[I] := ' '
   END;
     (* IDENTIFIER OR RESERVED WORD *)
  DES.SYMLOC := LOOKUP(BUFFER);
  IF DES.SYMLOC <> NIL THEN
   BEGIN
    IF DES.SYMLOC.KIND = RESWD THEN
     BEGIN
      NEXTSYM := DES.SYMLOC.KEY1;
      DES.LINEPOS := CC - 1
     END
    ELSE
     NEXTSYM := IDTOK
   END
  ELSE
   BEGIN
    NEXTSYM := IDTOK;
    DES.SYMLOC := ENTER(BUFFER)
   END;
  DES.SYMNAME := DES.SYMLOC.SYMNAME
 END; (* CASE OF 'A'..'Z'*)

'0','1','2','3','4','5','6','7','8','9' :
 BEGIN
  NEXTSYM := NUMTOK;
  BASE := 10;
  SCALE := 0;
  DECIMALFLAG := FALSE;
  BASEDNUMBER := FALSE;
  GETNUMBER(CH,BASE,DECIMALFLAG,NUMDIGITS,DES.INTVAL,DES.REALVAL);
  IF CH = ' ' THEN  (*BASED NUMBER*)
   BEGIN
    BASEDNUMBER := TRUE;
    BASE := DES.INTVAL;
    IF BASE > 16 THEN
     BEGIN
    ERROR(6,1,-1);
    BASE := 16
   END;
  INCHAR(CH);
  GETNUMBER(CH,BASE,DECIMALFLAG,NUMDIGITS,DES.INTVAL,DES.REALVAL);
  IF CH = '.' THEN
   BEGIN
    DECIMALFLAG := TRUE;
    DES.REALVAL := DES.INTVAL;
    DES.INTVAL := 0;
    DES.CHARVAL := 'R';
    INCHAR(CH);
    GETNUMBER(CH,BASE,DECIMALFLAG,NUMDIGITS,
              DES.INTVAL,DES.REALVAL);
    SCALE := -NUMDIGITS
   END;
  IF CH <> ' ' THEN
   ERROR(3,1,0);
  INCHAR(CH)
 END;
IF CH = '.' THEN (*DECIMAL PART *)
 BEGIN
  INCHAR(CH);
  IF BASEDNUMBER THEN
   ERROR(3,1,-1);
  DES.REALVAL := DES.INTVAL;
  DES.INTVAL := 0;
  DECIMALFLAG := TRUE;
  DES.CHARVAL := 'R';
  GETNUMBER(CH,BASE,DECIMALFLAG,NUMDIGITS,DES.INTVAL,DES.REALVAL);
  SCALE := -NUMDIGITS
 END;
IF CH = 'E' THEN (* EXPONENT PART *)
 BEGIN
  DECIMALFLAG := FALSE;
  DES.CHARVAL := 'R';
  IF SCALE = 0 THEN
   DES.REALVAL := DES.INTVAL;
  GETEXPONENT(SIGN,EXPONENT);
  IF SIGN THEN
   SCALE := SCALE - EXPONENT
  ELSE
   SCALE := SCALE + EXPONENT
 END;
IF SCALE <> 0 THEN  (*ADJUST SCALE *)
 BEGIN
  R := 1;
  SIGN := SCALE < 0;
  SCALE := ABS(SCALE);
  FAC := BASE;
  REPEAT
   IF ODD(SCALE) THEN
    R := R *FAC;
   FAC := SQR(FAC);
   SCALE := SCALE DIV 2
  UNTIL SCALE = 0;
  IF SIGN THEN
   DES.REALVAL := DES.REALVAL/R
  ELSE
   DES.REALVAL := DES.REALVAL * R
  END
 END; (* CASE OF '0'..'9' *)

'(',')','+',',','.',';',':','.','[',']' :
 BEGIN
  NEXTSYM := SPS[CH];
  DES.LINEPOS := CC;
  INCHAR(CH)
 END;

'"' : (* STRINGS *)
 BEGIN
  INCHAR(CH);
  ENDSTRING := FALSE;
  WHILE NOT ENDSTRING DO
   BEGIN
    WHILE CH <> '"' DO
     INCHAR(CH);
    INCHAR(CH);
    IF CH = '"' THEN
     INCHAR(CH)
    ELSE
     ENDSTRING := TRUE
   END;
  NEXTSYM := STRINGTOK
 END; (*STRINGS*)

'-' :
 BEGIN
  INCHAR(CH);
  IF CH = '-' THEN
   BEGIN
    INCHAR(CH);
    IF CH = ' ' THEN
     BEGIN
      INCHAR(CH);
      GETSYM(NEXTSYM,DES);
      IF NEXTSYM <> IDTOK THEN
       ERROR(17,1,-1)
      ELSE
       IF DES.SYMNAME = 'TRACEPARSE' THEN
        SWITCH[TRACEPARSE] := FLIP(SWITCH[TRACEPARSE])
       ELSE IF DES.SYMNAME = 'TRACETOK  ' THEN
        SWITCH[TRACETOK] := FLIP(SWITCH[TRACETOK])
       ELSE IF DES.SYMNAME = 'PRINTTABLE' THEN
        SWITCH[PRINTTABLE] := FLIP(SWITCH[PRINTTABLE])
       ELSE
        ERROR(17,1,-1)
     END;
    WHILE CC < LL DO
     INCHAR(CH);
    GETSYM(NEXTSYM,DES)
   END
  ELSE
   NEXTSYM := SPS['-']
 END;

'=' :
 BEGIN
  INCHAR(CH);
  IF CH = '>' THEN
   BEGIN
    NEXTSYM := ARROWTOK;
    INCHAR(CH)
   END
  ELSE
   IF CH = '=' THEN
    BEGIN
     NEXTSYM := EQUIVALENCE;
     INCHAR(CH)
    END
   ELSE
    NEXTSYM := SPS['=']
 END; (*CASE OF '=' *)

'*' :
 BEGIN
  INCHAR(CH);
  IF CH = '*' THEN
   BEGIN
    NEXTSYM := PWRTOK;
    INCHAR(CH)
   END
  ELSE
   NEXTSYM := SPS['*']
 END; (* CASE OF '*' *)

':' :
 BEGIN
  INCHAR(CH);
  IF CH = '=' THEN
   BEGIN
    NEXTSYM := BECOMESTOK;
    INCHAR(CH)
   END
  ELSE
   NEXTSYM := SPS[':']
 END; (* CASE OF ':' *)

```

```
          '/' :
           BEGIN
            INCHAR(CH);
            IF CH = '=' THEN
             BEGIN
              NEXTSYM := NOTEQTOK;
              INCHAR(CH)
             END
            ELSE
             NEXTSYM := SPS['/']
           END; (* CASE OF '/' *)

          '<' :
           BEGIN
            INCHAR(CH);
            IF CH = '=' THEN
             BEGIN
              NEXTSYM := LESSEQTOK;
              INCHAR(CH)
             END
            ELSE
             NEXTSYM := SPS['<']
           END; (* CASE OF '<' *)

          '>' :
           BEGIN
            INCHAR(CH);
            IF CH = '=' THEN
             BEGIN
              NEXTSYM := GTREQTOK;
              INCHAR(CH)
             END
            ELSE
             NEXTSYM := SPS['>']
           END; (* CASE OF '>' *)

          '''','$',' ','%','','?','●','','' :
           BEGIN
            ERROR(11,1,0);
            INCHAR(CH);
            GETSYM(NEXTSYM,DES)
           END
          END  (* CASE *)
         END; (* GETSYM *)



     PROCEDURE SEMANTIC (*PRODUCTION : INTEGER*);
      (* SEMANTIC AND SEMANTIC1 PERFORM THE STACK *)
      (* MANIPULATION BASED ON THE CURRENT PRO-   *)
      (* DUCTION NUMBER AND WILL EMIT THE PROPER  *)
      (* VALUES TO THE PRIMITIVE LIST, SYMBOL     *)
      (* TABLE AND TRANSLATE FILES                *)
```

```
VAR PTR : SYMPTR;
    TYPVAR : EXPTYPES;
    TEMPNAME : ALFA;
    PRCSN, I : INTEGER;
BEGIN
 CASE PRODUCTION OF
  2, 3,                     (* <AOP> ::= +/- *)
  4, 5,                     (* <MOP> ::= *// *)
  6, 7, 8, 9, 10, 11:       (* <RELATIONAL OP> ::= </<=/=/>/>=//= *)
   STACK[STKPTR].DES.INTVAL := PRODUCTION;

  12:                       (* <PRIMARY> ::= <NUMBER> *)
   IF STACK[STKPTR].DES.CHARVAL = '@' THEN    (*INTEGER*)
    BEGIN
     NEWCONS(STACK[STKPTR].DES.INTVAL, TEMPNAME,PRCSN);
     PUSHEVALSTACK(TEMPNAME, PRCSN);
     STACK[STKPTR].EXPTYPE := INT
    END
   ELSE                                        (* DES.CHARVAL = 'R' *)
    BEGIN
     ERROR(18,1,-3);
     PUSHEVALSTACK('JUNK      ',8);
     STACK[STKPTR].EXPTYPE := REEL
    END;

  13:                      (* <PRIMARY> ::= *STRING* *)
   BEGIN
    STACK[STKPTR].EXPTYPE := STNG;
    PUSHEVALSTACK('STRING    ',8)
   END;

  14:                      (* <PRIMARY> ::= <NAME> *)
   BEGIN
    WITH STACK[STKPTR].DES.SYMLOC DO
     CASE KIND OF TRANSDEC,ARITHMETIC,UNDEFINED:
      BEGIN
       IF KIND = TRANSDEC THEN
        PUSHEVALSTACK(SYMNAME,PRECISION6)
       ELSE
        IF KIND = ARITHMETIC THEN
         PUSHEVALSTACK(SYMNAME,PRECISION3)
        ELSE
         BEGIN
          PUSHEVALSTACK(SYMNAME,8);
          ERROR(28,1,-3)
         END;
       STACK[STKPTR].EXPTYPE := INT
      END;
      FNCTN:
       BEGIN
        IF TYPE8.SYMNAME <> 'ARITHMETIC' THEN
         ERROR(27,1,-3);
```

```
          PUSHEVALSTACK(SYMNAME,PRECISION8);
          STACK[STKPTR].EXPTYPE := INT
         END;
        OTHERWISE
         BEGIN
          ERROR(27,1,-3);
          PUSHEVALSTACK('JUNK       ',8);
          STACK[STKPTR].EXPTYPE := INT
         END
        END
       END;

    15:                            (* <PRIMARY> ::= ( <EXPRESSION> ) *)
     (* RETAIN INFORMATION ON EXPRESSION *)
     BEGIN
      STACK[STKPTR - 2] := STACK[STKPTR - 1]
     END;

    16:                ;  (* <FACTOR> ::= <PRIMARY> *)

    17:                ;  (* <FACTOR> ::= <FACTOR>**<PRIMARY> *)

    18:                ;  (* <TERM> ::= <FACTOR> *)

    19,                          (* <TERM> ::= <TERM> <MOP> <FACTOR> *)
    23,                (* <SIMPLE EXP> ::= <SIMPLE EXP> <AOP> <TERM> *)
    25:(* <SIMPLE EXP> ::= <SIMPLE EXP> <RELATIONAL OP> <SIMPLE EXP> *)
     BEGIN
      TYPVAR := CHECKTYPE(STACK[STKPTR - 2].EXPTYPE,
                         STACK[STKPTR].EXPTYPE);
      POPEVALSTACK(OPSTORE[3],SELSTORE[3]);
      POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
      SELSTORE[1] := COMPUTPRE(SELSTORE[2],SELSTORE[3]);
      NEWTEMP(SELSTORE[1],OPSTORE[1]);
      PUSHEVALSTACK(OPSTORE[1],SELSTORE[1]);
      CASE TYPVAR OF
       INT: CASE STACK[STKPTR-1].DES.INTVAL OF
        2 : (* + *) PUTS('add       ',3,3);
        3 : (* - *) PUTS('sub       ',3,3);
        4 : (* * *) PUTS('mult      ',3,3);
        5 : (* / *) PUTS('divide    ',3,3);
        6 : (* < *) PUTS('lt        ',3,3);
        7 : (* <= *) PUTS('le        ',3,3);
        8 : (* = *) PUTS('eq        ',3,3);
        9 : (* > *) PUTS('gt        ',3,3);
       10 : (* >= *) PUTS('ge        ',3,3);
       11 : (* /= *) PUTS('ne        ',3,3);
      END; (*CASE OF STACK*)
      STNG : ERROR(19,0,-1);
      BOOL : ERROR(21,0,-1);
      ERRORS : ERROR(20,0,-1);
     END;  (* CASE OF TYPEVAR *)
     IF STACK[STKPTR-1].DES.INTVAL IN [6..13] THEN
```

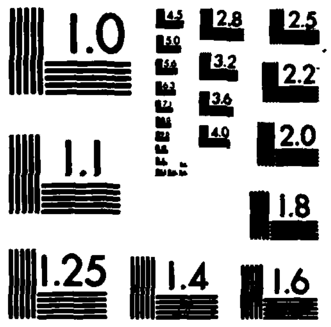
AD-A146 337   AN INPUT TRANSLATOR FOR A COMPUTER-AIDED DESIGN SYSTEM   2/2
(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA   T H CARSON
JUN 84

UNCLASSIFIED   F/G 9/2   NL

END

FILMED

DTIC


1.0
2.8
2.5
3.2
2.2
3.6
2.0
4.0
1.1
1.8
1.25
1.4
1.6

```
   STACK[STKPTR-2].EXPTYPE := BOOL;
 END;

20 :             (* <SIMPLE EXP> ::= <TERM> *)  ;

21 :             (* <SIMPLE EXP> ::= <AOP><TERM> *)
  IF STACK[STKPTR].EXPTYPE <> INT THEN
   ERROR(20,1,-1)
  ELSE
   IF STACK[STKPTR - 1].DES.INTVAL = 3 THEN  (*UNARY MINUS*)
    BEGIN
     POPEVALSTACK(OPSTORE[3],SELSTORE[3]);
     NEWCONS(0,OPSTORE[2],SELSTORE[2]);
     SELSTORE[1] := COMPUTPRE(SELSTORE[2],SELSTORE[3]);
     NEWTEMP(SELSTORE[1],OPSTORE[1]);
     PUTS('sub       ',3,3);
     PUSHEVALSTACK(OPSTORE[1],SELSTORE[1])
    END;

  22 :            (* <SIMPLE EXP> ::= NOT TERM *)
    IF STACK[STKPTR].EXPTYPE <> BOOL THEN
      ERROR(20,1,-1)
    ELSE
     BEGIN        (*GENERATE NOT*)
      POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
      NEWTEMP(SELSTORE[2], OPSTORE[1]);
             (* PRECISION, TEMPNAME *)
      PUSHEVALSTACK(OPSTORE[1],SELSTORE[1]);
      PUTS('not       ',2,2)
     END;

  24 :            (* <RELATION> ::= <SIMPLE EXP> *)  ;

  26 :            (* <EXP4> ::= <RELATION> *)       ;

  27,             (* <EXP4> ::= <EXP4> AND <RELATION> *)
  29,             (* <EXP3> ::= <EXP3> OR <EXP4> *)
  31,             (* <EXP2> ::= <EXP-2> => <EXP3> *)
  33 :            (* <EXPRESSION> ::= <EXPRESSION> == <EXP2> *)
    BEGIN
     TYPVAR := CHECKTYPE(STACK[STKPTR - 2].EXPTYPE,
                         STACK[STKPTR].EXPTYPE);
     POPEVALSTACK(OPSTORE[3],SELSTORE[3]);
     POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
     SELSTORE[1] := COMPUTPRE(SELSTORE[3],SELSTORE[2]);
     NEWTEMP(SELSTORE[1],OPSTORE[1]);
     PUSHEVALSTACK(OPSTORE[1],SELSTORE[1]);
     CASE TYPVAR OF
      BOOL : CASE PRODUCTION OF
       27 : PUTS('and       ',3,3);
       29 : PUTS('or        ',3,3);
       31 : PUTS('implicate ',3,3);
       33 : PUTS('equivalenc',3,3)
    END;
    INT : ERROR(23,1,-1);
    STNG : ERROR(19,1,-1);
    ERRORS : ERROR(20,1,-1)
   END
  END;

28 :              (* <EXP3> ::= <EXP4> *)    ;

30 :              (* <EXP2> ::= <EXP3> *)    ;

32 :              (* <EXPRESSION> ::= <EXP2> *)  ;

34 :              (* <EXPR LIST> ::=       *)
   STACK[STKPTR + 1].DES.INTVAL := 0;

35 :              (* <EXPR LIST> ::= <EXPRESSION> *)
   STACK[STKPTR].DES.INTVAL := 1;

36 :               (* <EXPR LIST> ::= <EXPR LIST>,<EXPRESSION> *)
   STACK[STKPTR-2].DES.INTVAL := STACK[STKPTR-2].DES.INTVAL + 1;

37 :          (* <IF THEN> ::= <IF HEAD> THEN <STMT GP> END IF *)
   BEGIN
    OPSTORE[1] := STACK[STKPTR - 4].DES.SYMNAME;
    PUTS('loc         ',1,0);
    PUTSYM('LOC        ',1,0)
   END;

38 :             (* <IF HEAD> ::= IF<EXPRESSION> *)
   BEGIN
    IF STACK[STKPTR].EXPTYPE <> BOOL THEN
     BEGIN
      ERROR(26,1,-1);
      STACK[STKPTR].EXPTYPE := BOOL
     END;
    POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
    NEWLABEL(OPSTORE[2]);
    PUTS('jmpf          ',2,1);
    STACK[STKPTR - 1].DES.SYMNAME := OPSTORE[2]
   END;

39 :       (* <WHILE DO> ::= <WHILE HEAD>DO<STMT GP>END WHILE *)
   BEGIN
    OPSTORE[1] := STACK[STKPTR -4].DES.SYMNAME;
    OPSTORE[2] := STACK[STKPTR -4].DES.TMPNAME;
    PUTS('whend       ',2,0)
   END;

40 :     (* <WHILE HEAD> ::= <WHILE><EXPRESSION>:<MAXLOOPCOUNT> *)
   BEGIN
    POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
    NEWLABEL(OPSTORE[2]);
    STACK[STKPTR - 3].DES.TMPNAME := OPSTORE[2];
    SELSTORE[1] := STACK[STKPTR].DES.INTVAL;
    PUTS('whilecon  ',2,1)
   END;

41 :                 (* <WHILE> ::= WHILE *)
   BEGIN
    NEWLABEL(OPSTORE[1]);
    STACK[STKPTR].DES.SYMNAME := OPSTORE[1];
    PUTS('whilestart',1,0)
   END;

42 :       (* <FOR LOOP> ::= <FOR HEAD>DO<STMT GP>END FOR *)
   WITH STACK[STKPTR - 4] DO
    BEGIN
     OPSTORE[1] := DES.SYMLOC.SYMNAME;
     SELSTORE[1] := DES.SYMLOC.PRECISION3;
     OPSTORE[2] := DES.SYMNAME;
     OPSTORE[3] := DES.TMPNAME;
     SELSTORE[2] := DES.INTVAL;
     PUTS('forend    ',3,2)
    END;

43 :(* <FORHEAD> ::= FOR *ID* FROM <EXPRESSION> TO *)
     (* <EXPRESSION> : <MAXLOOPCOUNT> *)
   BEGIN
    IF STACK[STKPTR - 6].DES.SYMLOC.KIND <> ARITHMETIC THEN
     ERROR(29,1,-1);
    OPSTORE[1] := STACK[STKPTR - 6].DES.SYMNAME;
    SELSTORE[1] := STACK[STKPTR - 6].DES.SYMLOC.PRECISION3;
    POPEVALSTACK(OPSTORE[3],SELSTORE[3]);
    POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
    NEWLABEL(OPSTORE[4]);
    NEWLABEL(OPSTORE[5]);
    SELSTORE[4] := STACK[STKPTR].DES.INTVAL;
    PUTS('forcons   ',5,4);
    STACK[STKPTR - 7].DES.SYMNAME := OPSTORE[4];
    STACK[STKPTR - 7].DES.TMPNAME := OPSTORE[5];
    STACK[STKPTR - 7].DES.SYMLOC := STACK[STKPTR - 6].DES.SYMLOC;
    STACK[STKPTR - 7].DES.INTVAL := SELSTORE[4]
   END;

44 :                     (* <PERFORM TASK> ::= *ID* *)
   BEGIN
    IF (STACK[STKPTR].DES.SYMLOC.KIND <> TASK) AND
       (STACK[STKPTR].DES.SYMLOC.KIND <> FNCTN) THEN
     ERROR(24,1,-1);
    OPSTORE[1] := STACK[STKPTR].DES.SYMNAME;
    PUTS('call      ',1,0)
   END;

45 : (* <PERFORM TASK> ::= <ID> ( <EXPR LIST> : <ID LIST> ) *)
   BEGIN
```

```
         ERROR(36,1,-1);
         IF (STACK[STKPTR - 5].DES.SYMLOC.KIND <> TASK) AND
            (STACK[STKPTR - 5].DES.SYMLOC.KIND <> FNCTN) THEN
          ERROR(24,1,STACK[STKPTR -4].DES.LINEPOS - CC - 3);
         OPSTORE[1] := STACK[STKPTR].DES.SYMNAME;
         PUTS('call      ',1,0)
        END;

      46 :                (* <MAXLOOPCOUNT> ::= <NUMBER> *)
        IF STACK[STKPTR].DES.CHARVAL <> '@' (*INTEGER*) THEN
         ERROR(13,1,-1);

      47 :                (* <LEFT PART LIST> ::= <NAME> := *)
        BEGIN
         TEMPLIST[1] := STACK[STKPTR - 1].DES.SYMLOC;
         TLI := 1
        END;

      48 :  (* <LEFT PART LIST> ::= <LEFT PART LIST> <NAME> := *)
        BEGIN
         TLI := TLI + 1;
         TEMPLIST[TLI] := STACK[STKPTR - 1].DES.SYMLOC
        END;

      49 : (* <ASSIGNMENT STATEMENT> ::= <LEFT PART LIST><EXPRESSION>*)
        BEGIN
         POPEVALSTACK(OPSTORE[2],SELSTORE[2]);
         FOR I := 1 TO TLI DO
          BEGIN
           OPSTORE[1] := TEMPLIST[I].SYMNAME;
           WITH TEMPLIST[I] DO
            CASE KIND OF
              BINARY: SELSTORE[1] := PRECISION2;
              ARITHMETIC: SELSTORE[1] := PRECISION3;
              TRANSDEC: SELSTORE[1] := PRECISION6;
              FNCTN: SELSTORE[1] := PRECISION8;
             OTHERWISE BEGIN
              ERROR(32,1,-1);
              SELSTORE[1] := 8
            END
           END
          END;
          PUTS('assign    ',2,2)
         END;

       50 :                       (* <DATA INPUT> ::= SENSE ( <NAME> ) *)
         WITH STACK[STKPTR - 1].DES.SYMLOC DO
          CASE KIND OF
            TRANSDEC : IF TYPE6.SYMNAME <> 'INPUT     ' THEN
                         ERROR(20,1,2)
                        ELSE BEGIN
                         OPSTORE[1] := SYMNAME;
                         SELSTORE[1] := PRECISION6;
```

```
                    PUTS('sensecond ',1,1)
                   END;
        UNDEFINED : ERROR(28,1,-2);
        OTHERWISE ERROR(20,1,-2)
     END;

51 :                    (* <DATA OUTPUT> ::= ISSUE (<NAME>) *)
  WITH STACK[STKPTR - 1].DES.SVMLOC DO
          CASE KIND OF
           TRANSDEC : IF TYPE6.SVMNAME <> 'OUTPUT    ' THEN
                        ERROR(20,1,-2)
                       ELSE BEGIN
                        OPSTORE[1] := SVMNAME;
                        SELSTORE[1] := PRECISION6;
                        PUTS('issuevent ',1,1)
                       END;
           OTHERWISE ERROR(20,1,-2)
         END;

52,53,54,55,56,57 :      (* <TIME MEASURE> ::= H/M/S/MS/US/NS *)
  STACK[STKPTR].DES.INTVAL := PRODUCTION;

58 :                  (* <PERIOD> ::= <NUMBER><TIME MEASURE> *)
  BEGIN
   IF STACK[STKPTR - 1].DES.CHARVAL = '@' THEN
    STACK[STKPTR-1].DES.REALVAL := STACK[STKPTR-1].DES.INTVAL;
   WITH STACK[STKPTR - 1] DO
     CASE STACK[STKPTR].DES.INTVAL OF
          (* CONVERT ALL TIMES TO MILLISECONDS *)
       52 : (*HOURS*) DES.REALVAL := DES.REALVAL * 3600000;
       53 : (*MINUTES*) DES.REALVAL := DES.REALVAL * 60000;
       54 : (*SECONDS*) DES.REALVAL := DES.REALVAL * 1000;
       55 : (*MILLISECONDS*);
       56 : (*MICROSECONDS*) DES.REALVAL := DES.REALVAL/1000;
       57 : (* NANOSECONDS*) DES.REALVAL := DES.REALVAL/1000000
     END
  END;

59 :                    (* <TIME> ::= <PERIOD> *)  ;

60 :                  (* <TIME> ::= <TIME><PERIOD> *)
  STACK[STKPTR-1].DES.REALVAL := STACK[STKPTR-1].DES.REALVAL +
       STACK[STKPTR].DES.REALVAL;

61 :(* <TIMED BLOCK> ::= <TIMEDBLOCKHEAD> ; <STMT GP>END IN *)
  PUTS('ni        ',0,0);

62 :              (* <TIMEDBLCKHEAD> ::= IN <PERIOD> *)
  BEGIN
   SELSTORE[1] := TRUNC(STACK[STKPTR].DES.REALVAL);
   PUTS('in        ',0,1)
  END;
  63 :             (* <WAIT> ::= WAIT <PERIOD> *)
    BEGIN
     SELSTORE[1] := TRUNC(STACK[STKPTR].DES.REALVAL);
     PUTS('fixedwait ',0,1)
    END;

  64 :             (* <WAIT> ::= WAIT <EXPRESSION> : <PERIOD> *)
    BEGIN
     POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
     SELSTORE[2] := TRUNC(STACK[STKPTR].DES.REALVAL);
     PUTS('waitleast ',1,2);
     IF STACK[STKPTR - 2].EXPTYPE <> INT THEN
       ERROR(38,1,STACK[STKPTR - 1].DES.LINEPOS - CC - 3)
    END;

  65 : (* <WAIT UNTIL> ::= <WAITHEAD><EXPRESSION> : <PERIOD> *)
    BEGIN
     POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
     OPSTORE[2] := STACK[STKPTR - 3].DES.SYMNAME;
     NEWLABEL(OPSTORE[3]);
     SELSTORE[2] := TRUNC(STACK[STKPTR].DES.REALVAL);
     PUTS('boolwait  ',3,2);
     IF STACK[STKPTR - 2].EXPTYPE <> BOOL THEN
      ERROR(26,1,STACK[STKPTR - 1].DES.LINEPOS - CC - 3)
    END;

  66 :              (* <WAITHEAD> ::= WAIT UNTIL *)
    BEGIN
     NEWLABEL(OPSTORE[1]);
     PUTS('stboolwait',1,0);
     STACK[STKPTR - 1].DES.SYMNAME := OPSTORE[1]
    END

  OTHERWISE SEMANTIC1(PRODUCTION) ;
 END (*CASE*)
END; (*SEMANTIC*)


PROCEDURE SEMANTIC1 (*PRODUCTION : INTEGER *);
  VAR PTR : SYMPTR;
      TYPVAR : EXPTYPES;
      TEMPNAME : ALFA;
      I : INTEGER;

  BEGIN
   CASE PRODUCTION OF

    86 :(*<INPUT SPEC> ::= INPUT : <TRANSMISSION BODY> END INPUT*)
       BEGIN

        FOR I := 1 TO TLI DO
         BEGIN
          TEMPLIST[I].TYPE6 := STACK[STKPTR - 4].DES.SYMLOC;
      OPSTORE[1] := TEMPLIST[I].SYMNAME;
      OPSTORE[2] := TEMPLIST[I].TECHNOLOGY6.SYMNAME;
      SELSTORE[1] := TEMPLIST[I].PRECISION6;
      PUTS('inputport ',2,1);
      PUTSYM('INPUTPORT',2,1)
     END;
    TLI := 0
   END;

87 :(* <OUTPUT SPEC> ::= OUTPUT :      *)
    (* <TRANSMISSION BODY> END OUTPUT *)
   BEGIN  (* ASSIGN TYPE = 'OUTPUT' *)
    FOR I := 1 TO TLI DO
     BEGIN
      TEMPLIST[I].TYPE6 := STACK[STKPTR - 4].DES.SYMLOC;
      OPSTORE[1] := TEMPLIST[I].SYMNAME;
      OPSTORE[2] := TEMPLIST[I].TECHNOLOGY6.SYMNAME;
      SELSTORE[1] := TEMPLIST[I].PRECISION6;
      PUTS('outputport',2,1);
      PUTSYM('OUTPUTPORT',2,1)
     END;
    TLI := 0
   END;

94 :    (* <DUPLEX SPEC> ::= DUPLEX :      *)
        (* <TRANSMISSION BODY> END DUPLEX *)
   BEGIN  (* ASSIGN TYPE = 'DUPLEX' *)
    FOR I := 1 TO TLI DO
     BEGIN
      TEMPLIST[I].TYPE6 := STACK[STKPTR - 4].DES.SYMLOC;
      OPSTORE[1] := TEMPLIST[I].SYMNAME;
      OPSTORE[2] := TEMPLIST[I].TECHNOLOGY6.SYMNAME;
      SELSTORE[1] := TEMPLIST[I].PRECISION6;
      PUTS('in/outport',2,1);
      PUTSYM('IN/OUTPORT',2,1)
     END;
    TLI := 0
   END;

95 :(* <BINARY SPEC> ::= BINARY : <BINARY BODY> END BINARY *);

96 :      (* <ARITHMETIC SPEC> ::=
          ARITHMETIC : <ARITHMETIC BODY> END ARITHMETIC *);

99 :      (* <TRANSMISSION DEC> ::= <ID> , <BINARY PRECISION> ,
                      <TECHNOLOGY> *)
   BEGIN
    PTR := STACK[STKPTR - 4].DES.SYMLOC;
    IF PTR.KIND <> UNDEFINED THEN
     ERROR(14,1,STACK[STKPTR - 3].DES.INTVAL - 1);
    PTR.KIND := TRANSDEC;
    PTR.PRECISION6 := STACK[STKPTR - 2].DES.INTVAL;
    PTR.TECHNOLOGY6 := STACK[STKPTR].DES.SYMLOC
```

```
  END;

102 :   (* <BINARY DEC> ::= <ID><STRUCTURE> , *)
        (* <BINARY PRECISION> <INITIAL VALUE> *)
 BEGIN
  PTR := STACK[STKPTR - 4].DES.SYMLOC;
  IF PTR.KIND <> UNDEFINED THEN
   ERROR(14,1,0);
  PTR.KIND := BINARY;
  PTR.PRECISION2 := STACK[STKPTR - 1].DES.INTVAL;
  IF STACK[STKPTR].DES.CHARVAL = '●' THEN
   PTR.IVAL2 := 0
  ELSE
   PTR.IVAL2 := STACK[STKPTR].DES.INTVAL
 END;

105 :     (* <ARITHMETIC DEC> ::= <ID><STRUCTURE> ,*)
          (* <DECIMAL PRECISION> <INITIAL VALUE>   *)
 BEGIN
  PTR := STACK[STKPTR - 4].DES.SYMLOC;
  IF PTR.KIND <> UNDEFINED THEN
   ERROR(14,1,0);
  PTR.KIND := ARITHMETIC;
  PTR.PRECISION3 := STACK[STKPTR - 1].DES.INTVAL;
  IF STACK[STKPTR].DES.CHARVAL = '●' THEN
   PTR.IVAL3 := 0
  ELSE
   PTR.IVAL3 := STACK[STKPTR].DES.INTVAL;
  OPSTORE[1] := PTR.SYMNAME;
  SELSTORE[1] := PTR.PRECISION3;
  SELSTORE[2] := PTR.IVAL3;
  PUTS('var        ',1,2);
  PUTSYM('VARIABLE   ',1,2)
 END;

106 :          (* <STRUCTURE> ::=     *);

107 :         (* <STRUCTURE> ::= ( <NUMBER LIST> )  *)
   ERROR(36,1,-1);

108 ,         (* <NUMBER LIST> ::= *NUMBER* *)

109 :      (* <NUMBER LIST> ::= <NUMBER LIST> , *NUMBER* *)
  BEGIN
   IF STACK[STKPTR].DES.CHARVAL <> '●' THEN
    ERROR(18,1,-1);
   NLI := NLI + 1;
   SELSTORE[NLI] := STACK[STKPTR].DES.INTVAL;
   IF PRODUCTION = 111 THEN
    STACK[STKPTR].DES.INTVAL := NLI
   ELSE
    STACK[STKPTR - 2].DES.INTVAL := NLI
  END;
```

```

110 :        (* <BINARY PRECISION> ::= <NUMBER>  *)
   IF STACK[STKPTR].DES.CHARVAL <> '●' THEN
    ERROR(13,1,-1);

111 :        (* <DECIMAL PRECISION> ::= <NUMBER> *)
   IF STACK[STKPTR].DES.CHARVAL <> '●' THEN
    ERROR(18,1,-1);

112 :         (* <INITIAL VALUE> *)
   STACK[STKPTR + 1].DES.CHARVAL := '●';

113 :         (* <INITIAL VALUE> ::= <NUMBER>  *)
   IF STACK[STKPTR].DES.CHARVAL = '●' THEN
    BEGIN
     STACK[STKPTR - 1].DES.CHARVAL := 'I';
     STACK[STKPTR - 1].DES.INTVAL := STACK[STKPTR].DES.INTVAL
    END
   ELSE
    ERROR(18,1,-1);

114,115,116 :      (* <TECHNOLOGY> ::= TTL/ECL/IIL *);

121 : (* <CODE SPEC> ::= CODE : *ID* , <BINARY PRECISION> ;
                        <CHARACTER REP LIST> END CODE *)
   BEGIN
    FOR I := 1 TO TLI DO
     TEMPLIST[I].CODID5 := STACK[STKPTR - 6].DES.SYMLOC;
    TLI := 0
   END;

97,98,118,119,122,123 :   (*  <A> ::= <B> , <A> ::= <A> <B> *)
  BEGIN
    TLI := TLI + 1;
    TEMPLIST[TLI] := STACK[STKPTR - 1].DES.SYMLOC
   END;

125,126,127,128,129 :(* <CODE ID> ::= <ID>/ASCII6/
                        ASCII7/EBCIDIC/BCD *);

130 :         (* <ID LIST> ::=      *)
   TLI := 0;

131 :         (* <IDLIST> ::= <ID> *)
   BEGIN
    TEMPLIST[1] := STACK[STKPTR].DES.SYMLOC;
    TLI := 1
   END;

132 :         (* <ID LIST> ::= <IDLIST> , <ID> *)
   BEGIN
    TLI := TLI + 1;
    TEMPLIST[TLI] := STACK[STKPTR].DES.SYMLOC
   END;

133 :       (* <NAME> ::= <ID> *);

134 :       (* <NAME> ::= <ID> ( <EXPRLIST> ) *)
   BEGIN
    STACK[STKPTR - 3].DES.CHARVAL := 'A' ; (*ARRAY*)
    ERROR(36,1,-1)
   END;

135 :
   BEGIN
    STACK[STKPTR - 5].DES.CHARVAL := 'B'; (*BIT FIELD*)
    ERROR(37,1,-1)
   END;

136 :   (* <FORMAL PARAM LIST> ::=      *)
   FIRSTPARAM := NIL;

137 :   (* <FORMAL PARAM LIST> ::= ( <ID LIST> : <ID LIST> ) *)
   ERROR(33,1,-1);

138 :     (* <PROC> ::= <TASK>  *);

139 :     (* <PROC> ::= <FUNCTION> *);

140,       (* <TASK> ::= <TASK HEAD> ; <ZOPT PROC DEC GP>
                      <STMT GP> END <ID> *)

146 : (* <FUNCTION> ::= <FUNCTION HEAD> ; <ZOPT PROC DEC GP>
                      <STMT GP> END <ID> *)
   BEGIN
    IF STACK[STKPTR - 5].DES.SYMLOC <> STACK[STKPTR].DES.SYMLOC
     THEN ERROR(25,1,-1);
    OPSTORE[1] := STACK[STKPTR - 5].DES.SYMNAME;
    PUTS('exitproc  ',1,0)
   END;

147,  (* <FUNCTION HEAD> ::= FUNCTION *ID* <FORMAL PARAM LIST>
                 BINARY , <BINARY PRECISION><INITIAL VALUE> *)

148 : (* <FUNCTION HEAD> ::= FUNCTION *ID* <FORMAL PARAM LIST>
            ARITHMETIC , <DECIMAL PRECISION><INITIAL VALUE> *)
   BEGIN
    PUTT(STACK[STKPTR - 6].DES.SYMNAME);
    OPSTORE[1] := STACK[STKPTR - 6].DES.SYMNAME;
    PUTS('proc      ',1,0);
    WITH STACK[STKPTR - 6].DES.SYMLOC DO
     BEGIN
      KIND := FNCTN;
      PARAMLIST8 := FIRSTPARAM;
      TYPE8 := STACK[STKPTR - 3].DES.SYMLOC;
      PRECISION8 := STACK[STKPTR - 1].DES.INTVAL;
```

```
          IVAL8 := STACK[STKPTR].DES.INTVAL
         END;
        STACK[STKPTR - 7] := STACK[STKPTR - 6]
       END;

145 :         (* <TASK HEAD> ::= TASK *ID* <FORMAL PARAM LIST> *)
       BEGIN
        PUTT(STACK[STKPTR - 1].DES.SYMNAME);
        OPSTORE[1] := STACK[STKPTR - 1].DES.SYMNAME;
        PUTS('proc        ',1,0);
        WITH STACK[STKPTR - 1].DES.SYMLOC DO
         BEGIN
          KIND := TASK;
          PARAMLIST7 := FIRSTPARAM
         END;
        STACK[STKPTR - 2] := STACK[STKPTR - 1]
       END;

156 :      (* <RANK> ::= <NU> *)
       STACK[STKPTR].DES.LINEPOS := STACK[STKPTR].DES.INTVAL;

157 :      (* <RANK> ::= <NU> . <PI> *)
       BEGIN
        STACK[STKPTR - 2].DES.LINEPOS :=
                   STACK[STKPTR - 2].DES.INTVAL;
        STACK[STKPTR - 2].DES.INTVAL := STACK[STKPTR].DES.INTVAL
       END;

158,    (* <NU> ::= <NUMBER> *)
159 :   (* <PI> ::= <NUMBER> *)
       IF STACK[STKPTR].DES.CHARVAL <> '●' THEN
        ERROR(13,1,-1);

160 :   (* <QUALIFICATION> ::=      *)
       STACK[STKPTR + 1].DES.CHARVAL := '●';

161 :         (* <QUALIFICATION> ::= IF <EXPRESSION> *)
       BEGIN
        STACK[STKPTR - 1].DES.CHARVAL := 'Q';
        IF STACK[STKPTR].EXPTYPE <> BOOL THEN
         ERROR(26,1,-1);
        POPEVALSTACK(OPSTORE[1],SELSTORE[1]);
        NEWLABEL(OPSTORE[2]);
        PUTS('jmpf        ',2,1);
        STACK[STKPTR - 1].DES.SYMNAME := OPSTORE[2]
       END;

162,163,164,165,166 :  (* <EPISODE TIMING> ::=    / : <ROE> /
                        : <ROE> , <B1> / : <ROE> , <B1> , <B2> /
                        : <ROE> , <B1> , <B2> , <RANK> *)
       BEGIN
        FOR I := 1 TO 5 DO
         SELSTORE[I] := 0;
```

```
      CASE PRODUCTION OF
        162 : ;
        163 : SELSTORE[1] := STACK[STKPTR].DES.INTVAL;
        164 : BEGIN
                SELSTORE[1] := STACK[STKPTR - 2].DES.INTVAL;
                SELSTORE[2] := STACK[STKPTR].DES.INTVAL
              END;
        165 : BEGIN
                SELSTORE[1] := STACK[STKPTR - 4].DES.INTVAL;
                SELSTORE[2] := STACK[STKPTR - 2].DES.INTVAL;
                SELSTORE[3] := STACK[STKPTR].DES.INTVAL
              END;
        166 : BEGIN
                SELSTORE[1] := STACK[STKPTR - 6].DES.INTVAL;
                SELSTORE[2] := STACK[STKPTR - 4].DES.INTVAL;
                SELSTORE[3] := STACK[STKPTR - 2].DES.INTVAL;
                SELSTORE[4] := STACK[STKPTR].DES.LINEPOS;
                SELSTORE[5] := STACK[STKPTR].DES.INTVAL
              END
             END
   END;

  167 :   (*<WHEN DO> ::= <QUALIFICATION> WHEN <NAME>
                          <EPISODE TIMING> DO <TASK LIST> *)

    BEGIN
     WITH STACK[STKPTR - 3].DES.SYMLOC DO
      CASE KIND OF
      UNDEFINED : ERROR(28,1,STACK[STKPTR -4].DES.LINEPOS-CC+2);
       BINARY : ;
       FNCTN : IF TYPE8.SYMNAME <> 'BINARY     ' THEN
                 ERROR(30,1,STACK[STKPTR-4].DES.LINEPOS-CC+2);
       OTHERWISE ERROR(30,1,STACK[STKPTR-4].DES.LINEPOS-CC)
      END;
     FOR I := 1 TO OPI DO
      PUTA(STACK[STKPTR-3].DES.SYMNAME,OPSTORE[I],5);
     IF STACK[STKPTR-5].DES.CHARVAL = 'Q' THEN
      BEGIN
       OPSTORE[1] := STACK[STKPTR-5].DES.SYMNAME;
       PUTS('loc       ',1,0);
       PUTSYM('LOC       ',1,0)
      END
    END;

 168 : (* <SIMPLE DO> ::= <QUALIFICATION> DO <TASK LIST> <RANK>*)

   BEGIN
    SELSTORE[1] := 0;
    FOR I := 1 TO OPI DO
     PUTA('          ',OPSTORE[I],5);
    IF STACK[STKPTR-3].DES.CHARVAL = 'Q' THEN
     BEGIN
```

```
          OPSTORE[1] := STACK[STKPTR-5].DES.SYMNAME;
          PUTS('loc        ',1,0)
         END
       END;

     169,    (* <EVERY> ::= <QUALIFICATION> EVERY <ROE>
                            DO <TASK LIST> *)

     170 :   (* <AT TIME> ::= <QUALIFICATION> AT <TIME>
                            DO <TASK LIST> *)

       BEGIN
        SELSTORE[1] := STACK[STKPTR-2].DES.INTVAL;
        FOR I := 1 TO OPI DO
         PUTA('            ',OPSTORE[I],5);
        IF STACK[STKPTR - 3].DES.CHARVAL = 'Q' THEN
         BEGIN
          OPSTORE[1] := STACK[STKPTR - 5].DES.SYMNAME;
          PUTS('loc        ',1,0)
         END
       END;


     171 :  (* <TASK LIST> ::= <NAME> *)

       BEGIN
        WITH STACK[STKPTR].DES.SYMLOC DO
         CASE KIND OF
          UNDEFINED : ERROR(28,1,-1);
          TASK : ;
          FNCTN : ;
          OTHERWISE ERROR(34,1,-1)
         END;
        OPSTORE[1] :=STACK[STKPTR].DES.SYMNAME;
        OPI := 1
       END;

     172 :  (* <TASK LIST> ::= <TASK LIST> THEN <NAME> *)

       BEGIN
        WITH STACK[STKPTR].DES.SYMLOC DO
         CASE KIND OF
          UNDEFINED : ERROR(28,1,-1);
          TASK : ;
          FNCTN : ;
          OTHERWISE ERROR(34,1,-1)
         END;
        OPI := OPI + 1;
        OPSTORE[OPI] := STACK[STKPTR].DES.SYMNAME
       END;

     182 :  (* <DESIGN CRITERIA> ::= DESIGN CRITERIA
                METRIC <METRIC> ; VOLUMES <NUMBER LIST> ;
```

```
                    MONITORS <NUMBER LIST> ; *)

             BEGIN
              PUTD(STACK[STKPTR-7].DES.SYMNAME,STACK[STKPTR-4].DES.INTVAL,
                   STACK[STKPTR-1].DES.INTVAL-STACK[STKPTR-4].DES.INTVAL);
              NLI := 0
             END;

          183,184,185 :  (* <METIRC> ::= FIRST/COST/POWER *);

          190 : (* PRINT ALL CONSTANTS USED DURING COMPILATION *)

             BEGIN
              PUTT('  SYSTEM  ');
              FOR I := 1 TO CSI DO
               BEGIN
                OPSTORE[1] := CONSTANTSTORE[I].NAME;
                SELSTORE[1] := CONSTANTSTORE[I].VAL;
                SELSTORE[2] := CONSTANTSTORE[I].PRECISION;
                PUTS('cons      ',1,2);
                PUTSYM('CONS       ',1,2)
               END;
              PRINTEMPS
             END;

          END   (*CASE PRODUCTION OF *)
          END;  (* PROCEDURE SEMANTIC1 *)

       BEGIN (*MAIN*)

        INITIALIZE;
        PUTT('   SYSTEM   ');
        PUTS('MAIN      ',0,0);
        PARSE;

        (*THE BELOW STMTS CHECK TO SEE IF TOGGLES HAVE*)
        (*BEEN TURNED ON TO TRACE THE PARSE EXECUTION *)
        (*AND CALLS THE PROCEDURE TO PRINT THE DETAILS*)

        99: IF LINERRPTR <> 0 THEN PRINTLINERRORS;
            IF PROGRAMERRFLAG THEN PRINTERRORS;
            IF SWITCH[PRINTTABLE] THEN GOPRINTTABLE
       END.
```

APPENDIX E

SCRATCH PAD EXAMPLE

```
CSDL TRANSLATOR                                    PAGE 1
NAVAL POSTGRADUATE SCHOOL
18-APR-19811:17:03.1
-- PRINTTABLE

IDENTIFICATION

   DESIGNER : "ALAN ROSS"
   DATE : "12-28-83"
   PROJECT : "DUAL PROCESS CONTROL APPLICATION"

DESIGN CRITERIA

  METRIC FIRST ;
  VOLUMES 8 ;
  MONITORS 8 ;


ENVIRONMENT

   INPUT : CONSIN,8,TTL ; CONST,8,TTL ; FLGA,1,TTL ;
           PINA,8,TTL ; FLGB,1,TTL ;
           PINB,8,TTL ; END INPUT ;

   OUTPUT : VA,8,TTL ; VB,8,TTL ; END OUTPUT ;

   ARITHMETIC : KCA,8 ; KCB,8 ; CNTB,8 ; ITIA,8 ; ITIB,8 ; AINT,8 ; TDA,8 ;
          TDB,8 ; BINT,8 ; VSA,8 ; VSB,8 ; BDIFF,8 ; PSA,8 ; PSB,8 ; CONPTT,8 ;
          EA,8 ; EB,8 ; KPIA,8 ; EA1,8 ; EA2,8 ; EB1,8 ; EB2,8 ; KPIB,8 ;
          END ARITHMETIC ;


PROCEDURES

   FUNCTION DATAA:
       BINARY ,1;
       SENSE (FLGA);
       IF FLGA = 1 THEN DATAA := 1; END IF ;
   END DATAA ;

    FUNCTION DATAB:
       BINARY ,1;
       SENSE (FLGB);
       IF FLGB = 1 THEN DATAB := 1; END IF ;
    END DATAB ;

   FUNCTION BCNT :
       BINARY ,1 ;
       IF CNTB >= 4 THEN BCNT := 1; END IF ;
   END BCNT ;

   TASK AFIX ;
       ARITHMETIC : ADIFF,8 ; END ARITHMETIC ;
```

```
1
CSDL TRANSLATOR                                     PAGE 2
NAVAL POSTGRADUATE SCHOOL
18-APR-19811:17:03.1
         SENSE (PINA);
         EA := PINA*KCA - PSA;
         ADIFF := (3*EA - 4*EA1 + EA2) *5;
         AINT := AINT + EA/KCA;
         VA := VSA + KCA*(EA + ITIA*AINT + TDA*ADIFF);
         ISSUE (VA);
         DATAA := 0;
         EA2 := EA1;
         EA1 := EA;
      END AFIX;

      TASK BCALC;
         SENSE (PINB);
         EB := PINB*KCB - PSB;
         BDIFF := (3*EB - 4*EB1 + EB2)*10;
         BINT := BINT + EB/KCB;
         CNTB := CNTB + 1;
         DATAB := 0;
      END BCALC;

      TASK BFIX;
         CNTB := 0;
         VB := VSB + KCB*(EB + ITIB*BINT + TDB*BDIFF);
         ISSUE (VB);
      END BFIX;

      FUNCTION CONFLG:
         BINARY,1;
         SENSE (CONSIN);
         IF CONSIN > 0 THEN CONFLG := 0; END IF;
      END CONFLG;

      TASK CHGCON;
         SENSE (CONST);
         IF CONPTT = 1 THEN KCA := CONST; END IF;
         IF CONPTT = 2 THEN ITIA := 1/CONST; END IF;
         IF CONPTT = 3 THEN TDA := CONST; END IF;
         IF CONPTT = 4 THEN VSA := CONST; END IF;
         IF CONPTT = 5 THEN PSA := CONST; END IF;
         IF CONPTT = 6 THEN AINT := CONST; END IF;
         IF CONPTT = 7 THEN KCB := CONST; END IF;
         IF CONPTT = 8 THEN ITIB := 1/CONST; END IF;
         IF CONPTT = 9 THEN TDB := CONST; END IF;
         IF CONPTT = 10 THEN VSB := CONST; END IF;
         IF CONPTT = 11 THEN PSB := CONST; END IF;
         IF CONPTT = 12 THEN BINT := CONST; END IF;
      END CHGCON;
```

```
  100 CONTINGENCY LIST
1
CSDL TRANSLATOR                                          PAGE 3
NAVAL POSTGRADUATE SCHOOL
18-APR-19811:17:03.1
  101         WHEN DATAA:100 MS DO AFIX;
  102         WHEN DATAB: 50 MS DO BCALC;
  103         WHEN BCNT: 100 MS DO BFIX;
  104         WHEN CONFLG DO CHGCON;
  105
  106     END
    1
   FUNCTION      RESWD         48
    2
   DATAB         FUNCTION   BINARY          1
   M             RESWD         57
    3
   DESIGN        RESWD         36
    4
    5
    6
   THEN          RESWD         72
    7
   PROJECT       RESWD         67
    8
    9
   BCNT          FUNCTION   BINARY          1
   TERM          RESWD         71
   10
   DATAA         FUNCTION   BINARY          1
   FOR           RESWD         46
   ARITHMETIC    RESWD         25
   11
   AND           RESWD         24
   12
   PRINTTABLE    UNDEFINED
   13
   EA            ARITHMETIC     8     0
   VSB           ARITHMETIC     8     0
   14
   EBCIDIC       RESWD         40
   CODE          RESWD         31
   15
   CHGCON        TASK
   VSA           ARITHMETIC     8     0
   VOLUMES       RESWD         78
   MS            RESWD         60
   16
   17
   UNTIL         RESWD         75
   18
   KCB           ARITHMETIC     8     0
   TTL           RESWD         74
  TASK          RESWD          70
  POWER         RESWD          65
  ISSUE         RESWD          55
19
  KCA           ARITHMETIC      8    0
  AT            RESWD          28
20
  BINT          ARITHMETIC      8    0
  PINB          TRANSDEC INPUT       TTL            8
  PINA          TRANSDEC INPUT       TTL            8
  WHILE         RESWD          81
  CONTINGENC    RESWD          32
21
  TDB           ARITHMETIC      8    0
  WHEN          RESWD          80
  EVERY         RESWD          44
  BINARY        RESWD          30
22
  TDA           ARITHMETIC      8    0
23
  BDIFF         ARITHMETIC      8    0
  IIL           RESWD          52
  BCD           RESWD          29
24
  EB            ARITHMETIC      8    0
25
  CONSIN        TRANSDEC INPUT       TTL            8
  DATE          RESWD          35
26
  AFIX          TASK
  TO            RESWD          73
27
  ECL           RESWD          41
28
  VA            TRANSDEC OUTPUT      TTL            8
  LIST          RESWD          56
29
30
  NOT           RESWD          61
31
  METRIC        RESWD          58
  IN            RESWD          53
32
  IDENTIFICA    RESWD          50
33
  S             RESWD          68
34
  DO            RESWD          38
35
  CNTB          ARITHMETIC      8    0
  MONITORS      RESWD          59
36
  FIRST         RESWD          45
  DUPLEX        RESWD        39
37
  PSB           ARITHMETIC    8    0
  ENVIRONMEN    RESWD        43
  ASCII7        RESWD        27
  ASCII6        RESWD        26
38
  CONPTT        ARITHMETIC    8    0
  PSA           ARITHMETIC    8    0
  ITIB          ARITHMETIC    8    0
  ITIA          ARITHMETIC    8    0
  CONST         TRANSDEC INPUT     TTL            8
  US            RESWD        76
  NS            RESWD        62
39
  VB            TRANSDEC OUTPUT    TTL            8
  OUTPUT        RESWD        64
40
  PROCEDURES    RESWD        66
41
  KPIB          ARITHMETIC    8    0
  KPIA          ARITHMETIC    8    0
  DESIGNER      RESWD        37
42
  EA2           ARITHMETIC    8    0
43
44
  EA1           ARITHMETIC    8    0
45
46
47
  FLGB          TRANSDEC INPUT     TTL            1
  FLGA          TRANSDEC INPUT     TTL            1
  WAIT          RESWD        79
  FROM          RESWD        47
  COST          RESWD        33
48
49
  BFIX          TASK
  BCALC         TASK
  H             RESWD        49
50
  VARIABLES     RESWD        77
  SENSE         RESWD        69
  END           RESWD        42
51
  AINT          ARITHMETIC    8    0
  OR            RESWD        63
52
  IF            RESWD        51
53
  CONFLG        FUNCTION   BINARY       1
  EB2           ARITHMETIC    8    0
  CRITERIA     RESWD        34
54
  ADIFF        ARITHMETIC    8    0
  EB1          ARITHMETIC    8    0
  INPUT        RESWD        54
```

APPENDIX F

SYMBOL TABLE

```
S. INPUTPORT(CONSIN,TTL:8)
S. INPUTPORT(CONST,TTL:8)
S. INPUTPORT(FLGA,TTL:1)
S. INPUTPORT(PINA,TTL:8)
S. INPUTPORT(FLGB,TTL:1)
S. INPUTPORT(PINB,TTL:8)
S.OUTPUTPORT(VA,TTL:8)
S.OUTPUTPORT(VB,TTL:8)
S.VARIABLE  (KCA:8,0)
S.VARIABLE  (KCB:8,0)
S.VARIABLE  (CNTB:8,0)
S.VARIABLE  (ITIA:8,0)
S.VARIABLE  (ITIB:8,0)
S.VARIABLE  (AINT:8,0)
S.VARIABLE  (TDA:8,0)
S.VARIABLE  (TDB:8,0)
S.VARIABLE  (BINT:8,0)
S.VARIABLE  (VSA:8,0)
S.VARIABLE  (VSB:8,0)
S.VARIABLE  (BDIFF:8,0)
S.VARIABLE  (PSA:8,0)
S.VARIABLE  (PSB:8,0)
S.VARIABLE  (CONPTT:8,0)
S.VARIABLE  (EA:8,0)
S.VARIABLE  (EB:8,0)
S.VARIABLE  (KPIA:8,0)
S.VARIABLE  (EA1:8,0)
S.VARIABLE  (EA2:8,0)
S.VARIABLE  (EB1:8,0)
S.VARIABLE  (EB2:8,0)
S.VARIABLE  (KPIB:8,0)
S.LOC       (@01:)
S.LOC       (@02:)
S.LOC       (@03:)
S.VARIABLE  (ADIFF:8,0)
S.LOC       (@04:)
S.LOC       (@05:)
S.LOC       (@06:)
S.LOC       (@07:)
S.LOC       (@08:)
S.LOC       (@09:)
S.LOC       (@10:)
S.LOC       (@11:)
S.LOC       (@12:)
S.LOC       (@13:)
S.LOC       (@14:)
S.LOC       (@15:)
S.LOC       (@16:)
S.CONS      (@C01:1,8)
S.CONS      (@C02:4,8)
S.CONS      (@C03:3,8)
S.CONS      (@C04:5,8)
S.CONS      (@C05:0,8)
```

```
S.CONS      (@C06:10,8)
S.CONS      (@C07:2,8)
S.CONS      (@C08:6,8)
S.CONS      (@C09:7,8)
S.CONS      (@C10:8,8)
S.CONS      (@C11:9,8)
S.CONS      (@C12:11,8)
S.CONS      (@C13:12,8)
```

LIST OF REFERENCES

1. Samish, F., "With Grand Designs," Micro Decisions (GB), No. 15 37-8, January 1983.

2. Ross, A.A., Computer Aided Design of Microprocessor-Based Controllers, Ph. D. Thesis, University of California, Davis, 1978.

3. Matelin, M.N., "Automating the Design of Dedicated Real Time Control Systems," VCRL-78651, Lawrence Livermore Laboratory, 21 August 1976.

4. Sherlock, B.J., User-Friendly Syntax Directed Input to a Computer Aided Design System, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1983.

5. Walden, H.J., The Application of a General Purpose Data Base Management System to Design Automation, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1983.

6. Barrett, W.A., Couch, J.D., Compiler Construction: Theory and Practice, Science Research Associates Inc., 1979.

7. Chomsky, N., "Three Models for the Description of Language", IEEE Transactions on Information Theory 2, 1956.

8. Shannon, A., The LR System. FORTRAN source listing for the LR system, National Energy Software Center, Version 61, Argonne, Illinois, 1979.

9. Wetherell, C. and Shannon, A., "LR, Automatic Parser Generator and LR(1) Parser.", Lawrence Livermore Laboratory, Livermore, California, 1979.

10. Myers, G.J., Composite Structured Design, Van Nostrand Reinhold company, New York, 1978.

11. Garlington, A.R., Preliminary Design and Implementation of an Ada Pseudo-Machine, M.S. Thesis, Air Force Institute of Technology, Dayton, 1981.

12. Dijkstra, E.W., "A Constructive Approach to the Problem of Program Correctness", BIT, Vol. 8, No. 3, 1968.

13. Howden, W. E., "Introduction to the Theory of Testing", _Tutorial: Software Testing & Validation Techniques_, IEEE Catalog No. EHO 138-8, 1978.

14. Miller, E., "Introduction to Software Testing Technology", _Tutorial: Software Testing & Validation Techniques_, IEEE Catalog No. EHO 138-8, 1978.

15. Howden, W.E., "Empirical Studies of Software Validation", _Tutorial: Software Testing & Validation Techniques_, IEEE Catalog No. EHO 138-8, 1978.

16. Matelan, M. N., "Automating the Design of Dedicated Real Time Control Systems", Lawrence Livermore Laboratory, Preprint UCRL-78651, 1976

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Dudley Knox Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
4.	Office of Research Administration Code 012A Naval Postgraduate School Monterey, California 93943	1
5.	Computer Technologies Curricular Office Code 37 Naval Postgraduate School Monterey, California 93943	1
6.	Thomas H. Carson 458 Papaya Court Jacksonville, Florida 32225	2
7.	LtCol. Alan A. Ross, USAF, Code 52Rs Department of Computer Science Naval Postgraduate School Monterey, California 93943	3